Trabajo Práctico Especial - RMI

Programación de Objetos Distribuidos

Grupo 6 - 2020Q2

Integrantes:

- *58639* Brula, Matías Alejandro
- *58546* de la Torre, Nicolás
- *59356* Tallar, Julián
- *59347* Vuoso, Julián Matías

A continuación se detallarán las decisiones y criterios tomados durante el desarrollo del trabajo práctico especial. Estos intentarán dar a entender los motivos detrás de las decisiones de diseño, implementación; criterios al momento de asegurar que el servidor sea Thread Safe, y las posibles mejoras a realizar.

Decisiones de Diseño e Implementación

En este inciso se mencionan las decisiones tomadas en el diseño de interfaces, implementaciones tanto del servant como de los clientes.

Para el control de la fiscalización mediante el *Callback Handler*, se incluyó un método adicional (*electionFinished*) por sobre el invocado por el servant para notificar el registro de un voto.
El método es utilizado para hacerle saber al cliente que ya no recibirá más votos, logrando así terminar de forma ordenada. Esto se logra particularmente llamando al método *unexportObject* de la clase *UnicastRemoteObject*.

En caso de capturar una *RemoteException* cuando se llama a un método del handler, se decide eliminar al handler del fiscal de la colección correspondiente.
El objetivo, es evitar sucesivos llamados a fiscales que ya no se encuentran disponibles. Cabe destacar, que el error podría ser recuperable y el handler se elimina indiscriminadamente.

En cuanto a la persistencia de los votos recibidos se tomaron dos decisiones principales.
La primera es la de almacenar todos los votos recibidos para una vez terminada la elección, procesarlos a medida que se solicita. Los resultados finales una vez calculados se guardan en memoria para atender los consiguientes pedidos a mayor velocidad.
Por otro lado, se almacenan los resultados parciales en memoria, en su formato FPTP, siendo estos actualizados a medida que se recibe un voto. De esta manera, ante la consulta parcial de los resultados, se puede responder con gran velocidad.

Finalmente para unificar los métodos de consultas se decidió agrupar las 3 clases de los tipos de resultado, STAR; SPAV; FPTP, bajo una misma clase abstracta padre: *Result*. De esta manera el cliente recibirá siempre un *Result*, que dependiendo del pedido realizado, y de si el mismo es parcial o no, *casteará* al tipo correspondiente.

Concurrencia y Criterios Aplicados

A continuación el cómo y por qué de la sincronización realizada en el servant para asegurar que este sea *thread safe*.

Se sincronizó con un mismo *lock* el **close**, **vote** (guardado de voto) y las 3 **queries** en caso de que se les deba responder con un resultado final. De esta manera aseguramos la coherencia total de los resultados y votos finales, evitando inconsistencias en las

respuestas a las consultas. Decidimos no utilizar el mismo lock para las instancias parciales para evitar esperas innecesarias, priorizando velocidad de procesamiento por sobre la constante coherencia de los resultados parciales (dado que, de todos modos, podrían cambiar de un momento a otro).

El método **open** no se sincronizó dado que todo conflicto terminaría siendo recuperable con la información parcial. De la misma manera el acceso al *status* en el **inspect** podría llegar a generar alguna inconsistencia, pero no sería de gran magnitud.

Dentro de la instancia de FPTP se sincronizaron los métodos de acceso al mapa de cómputo de votos. Se sincronizan aquí para no bloquear todo junto, nacional, provincial y mesa parcial.

Para los mapas con las provincias y tablas correspondientes, se sincronizó los **putIfAbsent**, dado que no es atómico, evitando así que que se sobreescriban y se vuelvan a crear los mapas.

Con un *lock* propio a lo largo del código del servant, se sincronizaron el *compute* y el *get* del **inspectorHandler**. Esto considera y abarca el caso en el que a un handler previamente eliminado, se le intente avisar nuevamente sobre un voto.

Con respecto a Threads y su implementación, se decidió utilizar un *FixedThreadPool* con 4 hilos para ejecutar las llamadas a métodos de los *Callback Handlers*. Así, el thread principal no debe esperar a que se establezca la conexión remota con cada cliente fiscal, sino que delega dicha tarea en los threads del pool.

Potenciales Mejoras

En primer lugar, con respecto a lo mencionado anteriormente de la remoción de fiscales, se podría tomar un criterio más laxo. Podríamos reintentar la llamada al método un tiempo después o bien permitirle varias posibles fallas previo a eliminarlo de la colección. También haría falta un testing con grandes volúmenes de datos para determinar adecuadamente el número de Threads convenientes para las llamadas a métodos de los *CallbackHandlers*.

Por otro lado, se podría agregar otro pool de threads para paralelizar el cómputo final de votos. Aunque este se realiza una única vez por tipo de consulta, podría agilizarse la velocidad de respuesta para el primer pedido.

Por último, un posible potencial inconveniente se daría si un cliente de management intenta cerrar la elección pero hay uno o más clientes de votación votando. No hay garantías de que, previo a ejecutarse el *close*, sigan entrando votos, ya que el synchronized no es por orden de llegada. Podría usarse algún *lock* con *fairness* para que sea más justo respecto al tiempo que lleva esperando cada thread, aunque eso también ralentizaría el proceso de votación.

Diagramas UML