Grupo 5: Nicolas Britos, Agustin Roca, Ignacio Grasso y Guido Barbieri

Informe:Trabajo práctico elecciones

Decisiones de diseño e implementación de los servicios:

Para los servicios decidimos armar cuatro interfaces remotas (una para cada cliente pedido) y tres clases que implementan las mismas: la clase de `VoteServiceImpl` implementa tanto esa interfaz como la interfaz `QueryService`, esto se debe a que es mejor mantener los resultados en una sola clase en lugar de tenerlos en dos diferentes (una de las cuales tendría una copia) debido a que habría que resolver concurrencia para modificar los resultados en la clase que mantiene la copia.

Por otro lado, armamos clases individuales para las distintas formas de contar los votos, clases que usa el servicio de votación para llevar el conteo. Por último, las clases que implementan interfaces remotas se agregan al registry en la clase server para poder ser accedidas remotamente.

Del lado del cliente armamos una clase con un main para cada interfaz (estas son las clases que llaman los distintos scripts) y que se conectan al servidor mediante RMI.

Criterios aplicados para el trabajo concurrente:

Para garantizar la consistencia de los datos, dentro de los métodos que registran un voto en las clases de cada sistema de votación (`FPTPVoting`, `STARVoting` y `SPAVVoting`) utilizamos algún tipo de mecanismo de bloqueo (`ReentrantLock` o método marcado como `synchronized`). También marcamos el método para iniciar o finalizar los comicios como `synchronized` para evitar que dos llamadas concurrentes al `open` no arroje errores.

Por otro lado, para los métodos que obtienen los resultados de las clases de sistemas de votación no fue necesario manejar la concurrencia, salvo en el `FPTPVoting` puesto que este es el único que puede ser consultado mientras los votos se van cargando, haciendo posible que la cantidad de votos registrada no coincida con el total si justo se agregó un voto al mismo tiempo. También fue necesario utilizar un `Lock` al momento de agregar un voto de una mesa en el `VoteServiceImpl` debido a que el mismo mantiene un mapa con todas las mesas, y si la mesa ingresada no existe es necesaria insertarla en el mapa para luego registrar un voto, y esto, si no se maneja concurrencia, puede llevar a pisar los votos de dicha mesa.

Además, decidimos crear un caché para los resultados de los votos con sistemas diferentes al `FPTP` para no tener que recalcularlos en cada llamada y, para

garantizar que estos resultados son computados, como máximo, una vez, fue necesario utilizar un `Lock` en el momento del cómputo de los mismos.

Finalmente, con la finalidad de registrar todos los votos que llegaron al servidor justo antes de cerrar los comicios y que aún no fueron procesados utilizamos un `Phaser`. La forma en que lo utilizamos se asemeja a una especie de semáforo invertido: por cada invocación al método `vote` del `VoteService` registramos al `Thread` actual en el `Phaser`, que lo podemos pensar cómo incrementar un contador. Una vez que finaliza la ejecución del método el mismo se desregistra, similar a decrementar el contador. Por otro lado, en el `AdminServiceImpl`, una vez que se llama al método `close` (para cerrar los comicios) se hace un `wait` en dicho `Phaser`, el cual bloquea la ejecución hasta que todos los `Threads` se hayan desregistrado (similar a pensar que retorna la ejecución una vez que el contador se vuelve cero). De esta manera evitamos que los comicios cierren y no se registren algunos votos y, por otro lado, logramos que el `QueryService` retorne los resultados de cada sistema de votación solamente cuando ya se terminaron de contar todos los votos, evitando retornar un resultado incorrecto.

Potenciales puntos de mejora y/o expansión:

Mejoras a futuro:

- Se podría agregar timeouts a todos los locks y medidas para mitigar la falla de ejecución de algún Thread.
- Se podría ampliar la cantidad de test de unidad y de integración.
- Se podría aumentar la cantidad de información logueada.
- Agregar comentarios en algunas clases en las que falta.