Informe Trabajo Práctico Final: Ruleta

1. Introducción

El presente trabajo consistió en desarrollar el juego Ruleta, el cual consiste en que varios usuarios dada una ronda o tirada, puedan realizar apuestas a un determinado número o categoría con un cierto monto, y luego obtener la ganancia/pérdida de dicha apuesta.

Además, este servicio debe poseer alta disponibilidad, es decir, que la caída de un nodo que ofrece el juego no perjudique ni comprometa el acceso, la jugabilidad ni la información que se procesa.

2. Solución

2.a) Arquitectura e implementación

Los actores esenciales involucrados en la solución son tres: **usuario**, **load balancer** y **ruleta**. El **usuario** va a poseer la interfaz de usuario por medio de una terminal (por simpleza del trabajo), para realizar apuestas correspondientes al servidor de juego. Después, quedará a la espera de una respuesta de ganancia o pérdida de sus apuestas, y una vez recibida estás, podrá volver a realizar el mismo flujo.

En segundo lugar, tenemos a la **ruleta** que es la encargada de ejecutar la tirada y procesar las apuestas de los usuarios, es decir, el pago o cobro de estas. La **ruleta** está representada por un conjunto de nodos, donde el **load balancer** va a seleccionar a una como **master** y las demás como **slaves**. Ambos roles van a recibir las apuestas de los usuarios a través del **load balancer** (no es casual esta decisión y se narrará de ello más adelante en la sección de problemáticas).

Por un lado, en la ruleta tenemos el rol del **master**, el cual se encarga de llevar la tirada y el procesamiento de las apuestas, y también, de replicar toda la información que vaya siendo procesada en cada instancia o estado en los **slaves**. Esto es debido a que este la posibilidad de que algún **slave** pueda tomar el lugar del **master** en cualquier momento. Por otro lado, los **slaves** además de recibir las apuestas de los usuarios y las réplicas del **master**, como se mencionó anteriormente. También, algún slave va a poder recibir un mensaje del **load balancer** adicional, en el caso de que tenga que tomar el rol de **master**, por el hecho de que se haya caído el nodo **master**.

Por último, el **load balancer** es el primer actor que obtenga las requests solicitadas por el **usuario** y hará de puente contra los nodos de **ruletas**. El rol que posee este actor es el de monitorear el estado de los nodos de **ruleta**, y notificar únicamente a algún nodo **slave** de **ruleta** para que tome el lugar de **master**, en caso de que se caiga el nodo **master**.

Se debatió en agregar más **load balancers**, para el caso en que este se caiga y no comprometa la disponibilidad del servicio. Pero, se llegó a concluir que esto es un problema de infraestructura, ya que se podría mitigar con el hecho de instanciar varios de estos actores de manera transparente y exponerlos en un mismo endpoint, por lo tanto el networking se encargaría de soportar dicha lógica.

Luego por una ventaja de secuenciar y unificar logs de las **ruletas**, se agregó a **loggy** (logger distribuido). Solo se agregó para los logs de los nodos de **ruleta**, ya que fue más sencillo sincronizar con los relojes Lamport, y nos otorgaba un verdadero valor a nivel de debug.

Más abajo en la **Figura A**, se encuentra un diagrama de la arquitectura explicada anteriormente, para facilitar la lectura e interpretación de esta.

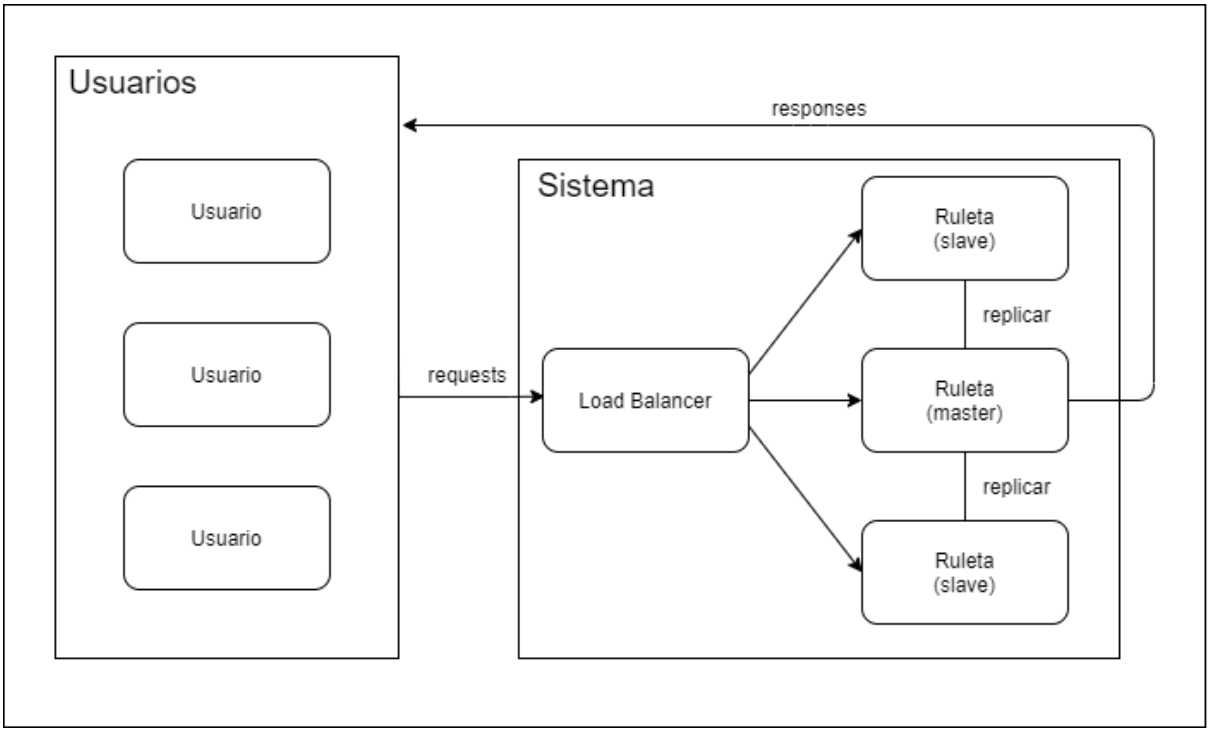

Figura A: Diagrama de arquitectura.

2.b) Problemáticas y alternativas

Una problemática que encontramos fue la de monitorear el PID junto al nodo, para poder tener un accionable frente a la caída del PID donde ejecuta algún servicio, en el caso que haya un error de lógica. Pero lo vimos demasiado difícil y no encontramos alguna función de erlang que nos provea dicha funcionalidad de manera accesible y limpia.

Luego, una de las principales problemáticas que nos encontramos fue la de hacer pruebas y debuggear correctamente. Esto nos llevó a agregar logs por cada interacción entre los actores que queríamos inspeccionar y/o analizar. Igualmente fue algo tedioso, pero junto a la ayuda de loggy pudimos cazar bugs que no vimos con tanta simpleza y claridad de manera separada. Uno de estos fue que en un principio el nodo master solo recibía las apuestas, es decir, se replicaban a los slaves cuando éste recibía los mensajes. Sin embargo, había un caso borde donde si el master recibía apuestas mientras procesaba las que ya tenía, y luego este se caía en ese momento, las apuestas que estaban encoladas se perdían. Para solucionar esto, como mencionamos en el apartado de arquitectura, tuvimos que enviar tanto al master como a los slaves las apuestas que se hacen. De esta manera, todos los nodos ruleta poseen la misma información, en el caso que el master ya no se encuentre disponible.

3. Conclusiones

En conclusión, creemos que hemos encontrado una solución aceptable ante la problemática de brindar alta disponibilidad del servicio del juego Ruleta. Algo que nos hubiese gustado agregar y no pudimos por falta de tiempo, es el poder sumar más nodos de ruleta, una vez que el servicio esté en funcionamiento. Sin embargo, creemos que no conlleva una gran dificultad, ya que solo bastaría con instanciar dicho nodo, y crear un mensaje que le mande al load balancer y a los peers de ruleta, para que este nodo nuevo se agregue en las listas de cada uno y continúe con su flujo normal.
También, hemos podido utilizar a loggy de un trabajo que vimos en clase para facilitarnos a la hora de debuggear y revisar logs de todos los nodos ruletas, ya sean slaves o master. Aunque esto tampoco fue fácil, ya que tuvimos que entender cuando debían incrementarse los tiempos lamport y que los tiempos de los nodos se sincronicen correctamente.
Por último, lo más complejo y tedioso del trabajo fue realizar las pruebas integrales en cada momento que se agregaba una funcionalidad o se corregía algún error, y luego analizarlas en caso de que no fueran exitosas. Sin embargo, nos vino muy bien para aprender a cómo enfrentarse con este tipo de arquitecturas y aprovechar algoritmos, patrones o ideas que vimos durante las clases para poder resolverlas de una manera óptima.