RETO 3 Y PLANTEAMIENTO DE RETO 4

La solución que propongo está basada en una DLT (Distribute Ledger Technology). En el *ledger* distribuido se encuentra toda la información de los tickets del sistema.

Profundizando un poco más, mandar toda su información cada cierto tiempo a todos los nodos conectados es, a todas luces, una solución ineficiente. Por lo tanto, cualquier mejora empieza reduciendo el flujo de información intercambiada.

Mi propuesta es transmitir solamente los cambios que se producen en la base de datos, no transmitir la base de datos al completo.

> Ya que la sincronización entre los nodos se hacía cada 5 minutos, se entiende que no hay necesidad de unos requisitos de tiempo de sincronización muy fuertes (en la versión anterior se tardaban cinco minutos en comunicar al resto del sistema un cambio).

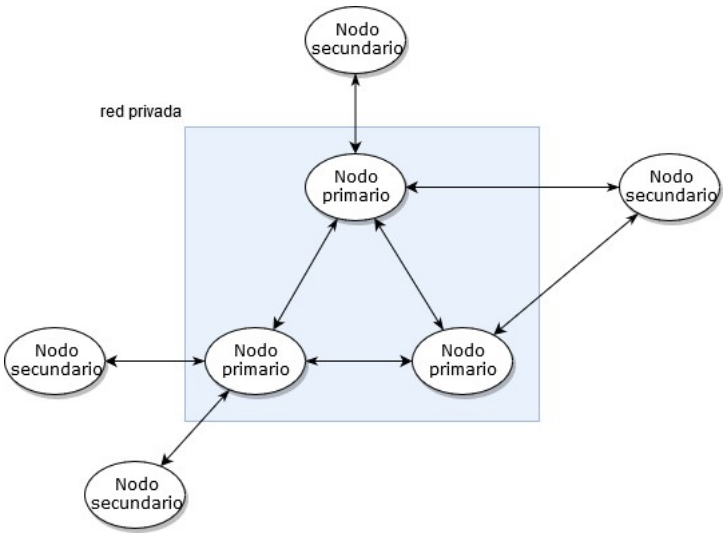
En la descripción del problema no hay mención a necesidad de soporte de conexión de nodos desconocidos ni a requisitos de seguridad y criptografía, por lo que podemos descartar la tecnología *blockchain*, por ser demasiado restrictiva en cuestión de rendimiento temporal y complejidad.

Tampoco se habla de una topología necesaria, descentralización o requisitos de resiliencia, por lo que concentrándonos en el rendimiento del sistema me he decidido por una red federada descentralizada formada por:

- Unos nodos principales que formarán la DLT y estarán ubicados en una red privada, como por ejemplo Hyperledger Fabric.

- Unos nodos secundarios, obligatoriamente conectados al menos a un nodo primario, que no mantienen una copia completa de la base de datos completa.

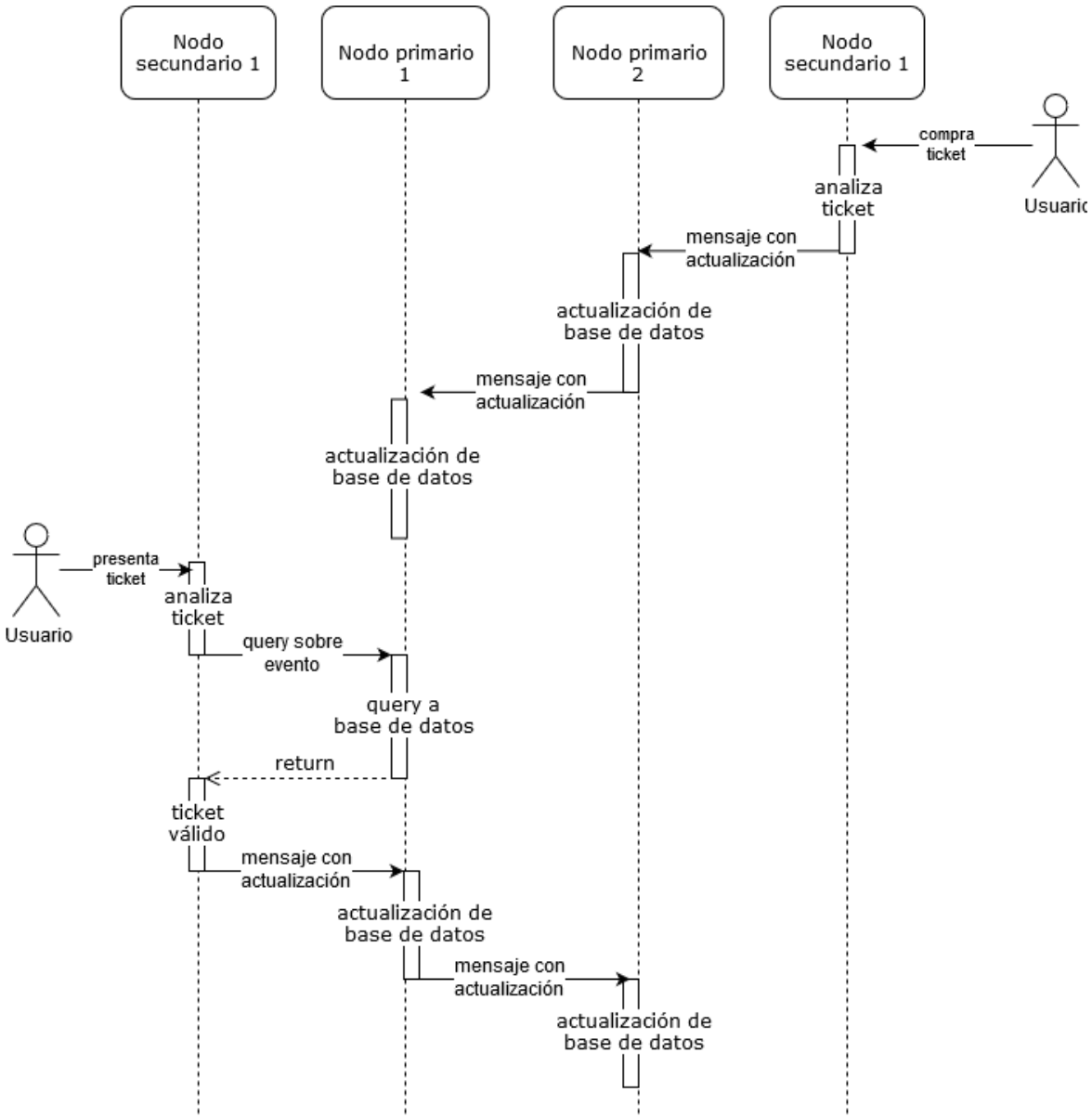
  Todos los nodos pueden recibir la consulta de lista de tickets usados, pero mientras que los nodos primarios hacen simplemente una consulta a su base de datos interna (la cual está sincronizada con el resto de nodos primarios), los nodos secundarios hacen esta consulta a los nodos primarios a los que están conectados y estos le devuelven el resultado.



Al alojar la DLT en una red privada, nos beneficiamos de la velocidad y seguridad asociadas a una red de este tipo. Las conexiones a los nodos secundarios no tendrían que ser de tanta calidad, ya que el flujo de datos sería mucho menor.

Aún así, una mejora simple a la propuesta sería un mecanismo de caché, en el que la consulta de un nodo secundario a un nodo primario incluyese el *timestamp* de la última. Si la *query* no devuelve un *timestamp* superior, el nodo primario envía un mensaje al nodo secundario indicando que su información del evento no está obsoleta. De esta manera, reduciríamos incluso más el flujo en los enlaces entre nodos secundarios y nodos primarios.

A continuación, se puede ver un diagrama de actividad muy sencillo con las acciones y conexiones entre los nodos.



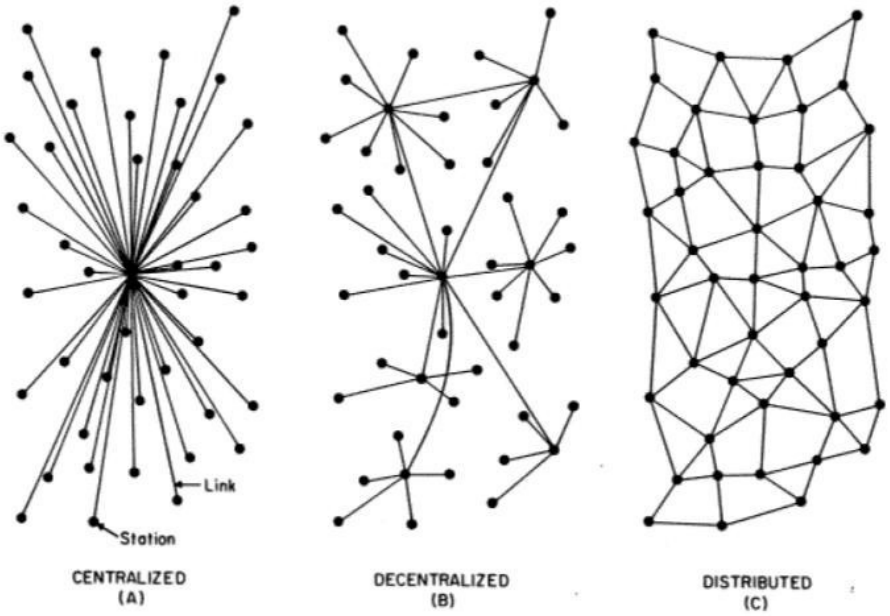
Puntos fuertes:

- Consumo de red es ínfimo comparado con la versión anterior, ya que solamente se intercambian los cambios en la base de datos y las consultas.

- El uso de una red privada para el alojamiento de la DLT permite que el sistema sea escalable.

Debilidades:

- La versión anterior del sistema tenía una topología completamente distribuida mientras que mi propuesta es solamente descentralizada (ver imagen siguiente). Esto la hace menos robusta a fallos de nodos. Sin embargo, creo que las ventajas asociadas en velocidad de proceso y ahorro de recursos compensan este inconveniente.



Proof of Concept

Para la POC, he optado por implementar la DLT con Hyperledger Fabric con CouchDB como base de datos.

He usado solamente un nodo primario, ya que la tecnología Hyperledger Fabric está suficientemente probada y madura, por lo que no es necesario su prueba.

He usado dos nodos secundarios que invocarán desde un script de bash los *chaincodes* de la instancia de Hyperledger Fabric (estos chaincodes están codificados en TypeScript) para actualizar y consultar la base de datos distribuida.

Tanto los chaincodes en TypeScript como los scripts de bash serán generados por Python a partir de ficheros JSON generados usando las soluciones a los retos 1 y 2.