UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Sistemas Distribuidos

Informe Laboratorio II

Integrantes: Hernán Aravena y Matías Barolo
Profesor: Manuel Ignacio Manríquez
Ayudante: Ariel Madariaga
Sección: A-1

Santiago – Chile
1-2024

ÍNDICE DE CONTENIDOS

1. Introducción

En el presente documento se detalla el procedimiento para la realización del laboratorio número 2 de la asignatura Sistemas Distribuidos, en el cual se realiza un análisis sobre un caso de estudio en relación a los conceptos de consistencia y replicación en sistemas distribuidos. Estos conceptos cobran vital importancia en un mundo donde los sistemas informáticos que operan en él son requeridos que sean cada vez más amplios.

Para afrontar estos desafíos inherentes a la envergadura de los sistemas deben crearse soluciones astutas que resuelvan problemas que surgen a partir de las condiciones por las que se rige la comunicación en los sistemas informáticos, como bien podría ser la latencia de acceso o tiempo de respuesta de algún sistema. Este tiempo estará acotado siempre por la velocidad a la que la electricidad viaja a través de los cables, por lo que para sistemas con alcance global se tendría en principio costo mínimo de comunicación. Es en este punto donde los conceptos como la replicación entran. En vez de confiar en un punto de acceso a un sistema localizado geográficamente en un solo lugar, se puede optar por ofrecer el mismo sistema a través de servidores localizados estratégicamente para reducir este tipo de costo.

Asimismo, debido a la masificación del uso de dichos sistemas y la integración de ellos es necesario tomar los resguardos necesarios que implica el acceso concurrente a estos de manera de evitar las discrepancias entre datos de un mismo o varios sistemas.

2. Caso de estudio

En lo que respecta a sistemas distribuidos, Google es un exponente que claramente representa al tema. Tratándose de un motor de búsqueda web que debe funcionar prácticamente en todo el mundo y con tiempos de respuesta bajísimos es claro que existe una arquitectura e infraestructura detrás que está a la altura del desafío, tal como así son algunas de las herramientas que ofrece como Google Docs y Google Drive, que permite la edición colaborativa de documentos en línea de manera instantánea y el almacenamiento de archivos en la nube, respectivamente. Estas herramientas en particular son el objeto de estudio de este trabajo, ya que nos permiten analizar de buena manera los conceptos antes mencionados de consistencia y replicación.

Primero, para permitir la correcta visualización de los distintos documentos de los que se disponga en la herramienta y soportar la edición para el número de usuarios que se requiera se debe contar con un modelo de consistencia de los datos que se las brinde ante este tipo de uso. Por otra parte, el acceso a estos documentos alojados en la nube no tienen por qué estar limitados por la región en la cual el cliente se encuentre geográficamente, por lo que la arquitectura debe contar también con las políticas de replicación que correspondan para asegurar que el acceso a los datos sea posible a través de cualquier parte del mundo, a un tiempo razonable, también sea dicho.

3. Arquitectura propuesta

Centrándonos en el funcionamiento concurrente de Google Docs, en su funcionamiento fundamental con el propósito de simplificarlo podríamos notar sólo dos operaciones básicas, se agrega un carácter al documento, o se elimina un carácter al documento en la posición en la que se encuentra el cursor del texto.

Es así entonces que inicialmente podría proponerse un modelo de consistencia el cual involucra incrementos de actualización al documento. Estos incrementos consistirían sólo de la adición o eliminación de un carácter del documento (obviando el comportamiento del cursor del texto) y podrían ser tratados en una primera instancia de manera meramente local. Para mantener la consistencia del documento en los casos que existan múltiples usuarios accediendo al documento estos deberían “encolarse” para ser enviados al servidor, el cual iría recibiendo dichos incrementos y realizando el cambio que sea pertinente a la “copia” del documento de la nube. Como puede darse el caso de que la copia local del documento de un cliente aún tenga cambios por enviar al servidor, puede darse el caso de que al actualizar el documento este de repente pierda trabajo realizado por

el cliente (ya que los cambios aún no eran enviados), de manera que el servidor deberá también informar a los demás clientes del cambió que fué realizado para que este también pueda procesar dicho cambio en su copia del documento. De esta manera entonces se podría mantener la consistencia del documento entre los clientes.

Debido a lo anterior, debe elegirse entonces un modelo de consistencia de datos datacéntricos, esto para asegurar la consistencia del documento entre sin importar el número de clientes. En particular una buena elección para la arquitectura antes mencionada podría bien ser la consistencia secuencial, esto debido a que cada actualización del documento será procesada por el servidor de manera atómica, por lo que cuando este reciba un cambio en el documento deberá informar a los clientes para que estos lo incluyan también en su copia local. Lo anterior debe acompañarse también de cuidar que las operaciones se realicen en el menor tiempo posible, ya que si los clientes demoran en recibir las actualizaciones los usuarios bien podrían tener problemas de coordinación a la hora de realizar cambios en el documento.

En la figura 1 se ilustra la arquitectura propuesta, cuando se realiza un cambio en el documento, el servidor reenvía el cambio a los clientes para que rectifiquen el documento, esto en la práctica debe hacerse también cuidando la posición en la que se envía dicho cambio de manera de que los clientes puedan introducirlos en la copia local en el lugar del documento donde efectivamente debe hacerse la modificación.

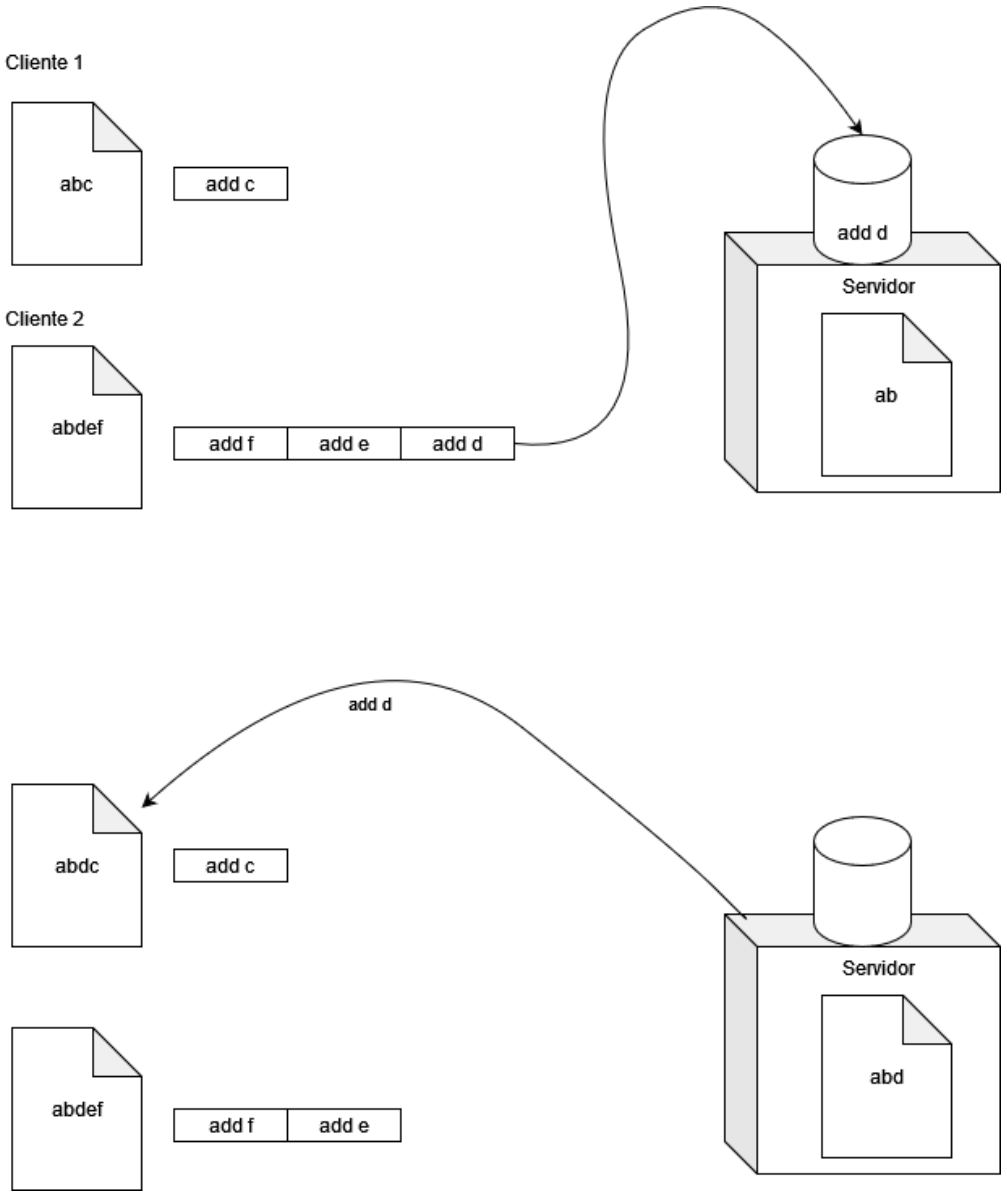

Figura 1: Arquitectura de consistencia

4. Metodología

Con el objetivo de modelar la situación planteada, es decir, el comportamiento de la consistencia y la replicación en un sistema distribuido como lo es Google Docs, se utilizarán Apache Kafka y Apache Spark.

El diseño del sistema tiene una arquitectura general en donde ambas tecnologías de Apache serán utilizadas de la siguiente manera:

Apache Kafka será utilizado para la gestión de mensajes en tiempo real,

permitiendo la transmisión de actualizaciones de documentos entre diferentes nodos del sistema.
Apache Spark se empleará para procesar y aplicar estas actualizaciones de manera eficiente, asegurando que todas las copias de los documentos en los diferentes nodos permanezcan consistentes.

Principales componentes:

Productores (Clientes de Google Docs):

Los clientes que interactúan con el documento (añadiendo o eliminando caracteres) actúan como productores en Apache Kafka, enviando eventos de actualización.

Kafka Topics:

Cada documento puede estar asociado a un *topic* en Kafka, donde se publican todas las actualizaciones de ese documento.

Spark Streaming:

Apache Spark recibirá los datos en tiempo real desde Kafka y aplicará las transformaciones necesarias para mantener la consistencia del documento.

Consumo y Aplicación de Cambios:

Se Implementará un clúster de Kafka para gestionar la transmisión de mensajes entre los diferentes nodos. Se configurarán los tópicos para cada documento que se replique con el fin de asegurar la consistencia.

5. Resultados y discusión

La implementación realizada aseguró que el paso de mensajes (tópicos) fuera realizado apropiadamente entre el programa producer.py (donde se definen los clientes que envían mensajes) y el programa consumer.py (servidor). Se asegura la consistencia secuencial debido a que los mensajes son guardados y mostrados en el orden que llegan.

Por otra parte, no fue posible implementar las actualizaciones de las copias del documento de los clientes debido a problemas con el manejo del tiempo. Para la implementación de aquello y en retrospectiva podría haberse optado por implementar una suerte de productor-consumidor, de manera de simular de manera más fiel el comportamiento descrito en la arquitectura propuesta. De esta forma, el

servidor podría enviar también las actualizaciones correspondientes y los clientes recibirlas para realizar los cambios en sus respectivas copias del documento.

Un ejemplo de la ejecución de producer y consumer sería el siguiente

```
C:\Users\Matix\OneDrive\Escritorio\lab2distri\Lab2-Sistemas-Distribuidos>python producer.py
Client 1 is sending: {'client': 'Client 1', 'change': {'type': 'remove', 'char': 'r', 'pos': 0}, 'position': 2, 'timestamp': 1724632982}
Client 1 is sending: {'client': 'Client 1', 'change': {'type': 'remove', 'char': 'i', 'pos': 0}, 'position': 10, 'timestamp': 1724632982}
Client 1 is sending: {'client': 'Client 1', 'change': {'type': 'remove', 'char': 'j', 'pos': 0}, 'position': 6, 'timestamp': 1724632982}
Client 2 is sending: {'client': 'Client 2', 'change': {'type': 'remove', 'char': 'u', 'pos': 0}, 'position': 7, 'timestamp': 1724632982}
Client 2 is sending: {'client': 'Client 2', 'change': {'type': 'add', 'char': 't', 'pos': 0}, 'position': 4, 'timestamp': 1724632982}
Client 1 is sending: {'client': 'Client 1', 'change': {'type': 'add', 'char': 'v', 'pos': 0}, 'position': 6, 'timestamp': 1724632982}
Client 2 is sending: {'client': 'Client 2', 'change': {'type': 'add', 'char': 'p', 'pos': 0}, 'position': 10, 'timestamp': 1724632982}
Client 2 is sending: {'client': 'Client 2', 'change': {'type': 'remove', 'char': 'i', 'pos': 0}, 'position': 1, 'timestamp': 1724632982}
Client 2 is sending: {'client': 'Client 2', 'change': {'type': 'add', 'char': 'v', 'pos': 0}, 'position': 0, 'timestamp': 1724632982}
Client 2 is sending: {'client': 'Client 2', 'change': {'type': 'add', 'char': 'o', 'pos': 0}, 'position': 0, 'timestamp': 1724632982}
```

Figura 2: Ejemplo de ejecución producer.py

```
C:\Users\Matix\OneDrive\Escritorio\lab2distri\Lab2-Sistemas-Distribuidos>python consumer.py
Copia del documento en servidor:
Copia del documento en servidor:
Copia del documento en servidor:
Copia del documento en servidor:
Copia del documento en servidor: t
Copia del documento en servidor: tv
Copia del documento en servidor: tvp
Copia del documento en servidor: tv
Copia del documento en servidor: tvv
Copia del documento en servidor: tvvo
```

Figura 3: Ejemplo de ejecución consumer.py

6. Conclusiones

Si bien durante el transcurso del trabajo algunos objetivos, como se mencionó en la sección anterior, no pudieron ser completados como la comunicación bidireccional en la simulación en general los objetivos teóricos del laboratorio sí fueron cumplidos, ya que pudo analizarse de manera satisfactoria el caso de estudio escogido y además pudo ser propuesta una arquitectura que, guardando las proporciones, podría resolver algunos de los desafíos propuestos por el caso de estudio

a. Limitaciones

En su mayor parte las limitaciones para la realización de este trabajo fueron en cuanto a la poca familiaridad que se tiene respecto a las herramientas utilizadas, en este caso Apache Spark y Apache Kafka, pese a esto de igual forma pudieron ser resueltas porciones de la parte práctica del laboratorio. Por otra parte, el manejo del tiempo supuso una limitación pues esto fue el impedimento más grande a la hora de completar el trabajo.

b. Trabajo Futuro

En cuanto al trabajo futuro entonces, queda la implementación completa de la arquitectura propuesta, con el envío bidireccional de mensajes entre el servidor y clientes. Por otra parte, una adición interesante a este trabajo sería la implementación de la consistencia a través de un sistema que involucre el cálculo de diferencias a partir de las distintas copias que puedan existir en una serie de

clientes y el servidor de un documento, algo similar a lo que hace Github con su sistema de versionamiento de código, de manera de ofrecer mejor consistencia de datos al sistema sobre todo en casos donde se utilice por una cantidad considerable de usuarios.