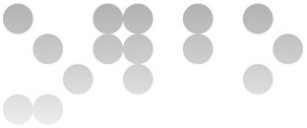
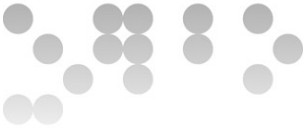
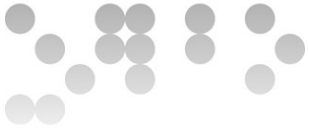


Sistemas Distribuidos

PEC 1 - El reto de los sistemas informáticos a escala Internet



Índice



Pregunta 1

Enunciado

Explicar el concepto de transparencia en sistemas distribuidos. Proporcionar ejemplos de transparencia de acceso y transparencia de escala.

Respuesta

La transparencia es la ocultación, tanto al usuario como al programador de aplicaciones, de la separación de los diferentes componentes de un sistema distribuido, provocando que estos se muestren como un todo en lugar de una colección independiente de componentes.

De igual forma, la transparencia oculta y hace anónimos los recursos irrelevantes para la tarea que estén realizando, de nuevo, los usuarios y programadores de aplicaciones.

Hay identificadas 8 tipos de transparencia, según el manual de referencia de la ANSA (*Advanced Networked Systems Architecture*) y de la RM-ODP (International Organization for Standardization's Reference Model for Open Distributed Processing)

Estos 8 tipos de transparencia son: acceso, localización, concurrencia, replicación, fallos, movilidad, rendimiento y escala.

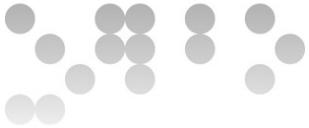
La transparencia de acceso permite acceder, mediante operaciones idénticas, a los recursos locales y remotos del sistema. Como ejemplos de esta tendríamos:

- Una única interfaz gráfica de usuario con carpetas, tanto para archivos locales como remotos

- Una aplicación para archivos que utiliza las mismas operaciones para acceder tanto a archivos locales como remotos.

La transparencia de escala es la que permite que se amplié en escala el sistema y las aplicaciones, sin cambios para la estructura del sistema o los algoritmos de la aplicación.

Un ejemplo de escalabilidad seria que una aplicación web, cuyo transito habitual no pasara de varias decenas de usuarios, recibiera de golpe, y sin ver su rendimiento afectado, un pico de varios miles de usuarios.



Pregunta 2

Enunciado

Explicar las diferencias entre el escalado vertical y horizontal en sistemas distribuidos con ejemplos.

Respuesta

Ampliando la definición de **escala** del apartado anterior, **la escalabilidad** se refiere a la capacidad de un sistema para brindar un rendimiento razonable bajo un incremento de la demanda, o por aumento de tráfico o por aumento de volumen de datos. Un sistema escalable debe funcionar bien bajo una carga creciente, además de reducir la necesidad de tener que rediseñar el sistema ante tales desafíos.

Sabiendo esto, hay 2 tipos posibles de escalado, el escalado vertical y el horizontal.

El escalado vertical, o escalado hacia arriba, consiste en maximizar los recursos de un solo equipo o nodo de un sistema, con el objetivo de aumentar su capacidad para manejar una carga creciente. Aunque la ampliación puede ser relativamente sencilla, este método adolece de varias desventajas.

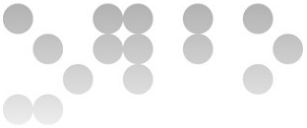
El principal es que el coste de la expansión aumenta de forma exponencial, eso se debe a que el aumento de coste de un mejor hardware no es proporcional a el aumento de potencia que proporciona. Igualmente, una vez tienes el mejor hardware disponible, es imposible seguir con este escalado.

Por último, existe el requisito de tiempo de inactividad para escalar. Si todos los servicios y datos de la aplicación web residen en una sola unidad, la escala vertical en esta unidad no garantiza la disponibilidad de la aplicación durante dicho escalado. De la misma forma, si solo existe una sola máquina, si esta llegara a fallar, la aplicación estaría inoperativa.

El escalado horizontal se refiere al incremento de recursos mediante la adición de equipos o nodos al sistema, lo que significa agregar más unidades al sistema en lugar de incrementar la capacidad de una sola unidad, como se hace en el escalado vertical. Las solicitudes de recursos y/o peticiones se distribuyen luego entre estas unidades.

El tener un sistema con múltiples nodos permite mantener el sistema operativo, aunque se caiga uno o varios de ellos. Además, normalmente el coste de tener varias máquinas más pequeñas es menor que el costo de una sola unidad de mayor calidad. Por tanto, la escala horizontal suele ser más rentable que el escalado vertical.

Sin embargo, también existen desventajas en el escalado horizontal. Aumentar el número de unidades significa que es necesario invertir más recursos en su mantenimiento.



Además, el código de la aplicación en sí debe modificarse para permitir el paralelismo y la distribución del trabajo entre varias unidades. En algunos casos, esta tarea no es trivial y escalar horizontalmente puede ser una tarea difícil.

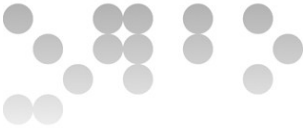
Vamos a comparar ambos mediante un **ejemplo**.

Disponemos de una aplicación web alojada en un cluster de balanceo de dos nodos, que se reparten la carga de las peticiones recibidas.

A lo largo del año se ha aumentado el volumen de las peticiones recibidas, llegando hasta el limite de ambos servidores, y se requiere de una ampliación de este.

Un ejemplo de ampliación mediante escalado horizontal seria agregar potencia de procesamiento y de RAM a las maquinas físicas donde está configurado cada nodo.

Por último, un ejemplo de ampliación de escalado horizontal seria aumentar el número de nodos en el cluster de balanceo, añadiendo mas maquinas con la misma capacidad que las dos ya existentes



Pregunta 3

Enunciado

Comparar modelos de comunicación síncrona y asíncrona con diferentes ejemplos reales. Explica cuándo usar cada uno de ellos. ¿Qué modelo es más escalable? ¿Por qué?

Respuesta

Cuando hablamos de comunicación síncrona y asíncrona nos estamos refiriendo a dos formas de intercambio de información en función de la simultaneidad con la que se envía y ofrece el mensaje.

La comunicación síncrona tiene las siguientes características:

- El emisor envía datos al receptor y queda a la espera de una respuesta antes de continuar o finalizar la comunicación.

- Tanto emisor como receptor deben estar activos simultáneamente durante la comunicación.

- Cuando se ejecuta una comunicación síncrona hay que esperar a que finalice antes de poder realizar una nueva acción.

La comunicación asíncrona tiene las siguientes características:

- El emisor envía una comunicación a una cola gestionada por el receptor y no se requiere de una respuesta inmediata de este.

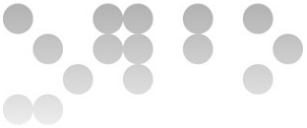
- El receptor no necesita estar activo durante la comunicación, ya que recibirá la misma una vez pase a estar activo.

- Se pueden seguir realizando acciones tras ejecutar una comunicación asíncrona, sin necesidad de esperar respuesta.

Viendo las características de ambas, podemos concluir que se debe utilizar la comunicación síncrona cuando sea necesaria una respuesta inmediata y en tiempo real a la comunicación realizada por el emisor.

Cuando no es necesaria una respuesta inmediata, pueden utilizarse ambos tipos de comunicaciones, pero la más recomendable en este caso es la asíncrona, puesto que es la más adecuada y la que consume menos recursos.

Vamos a comparar ambos tipos de comunicaciones utilizando como **ejemplos** aplicaciones de mensajería o comunicación.

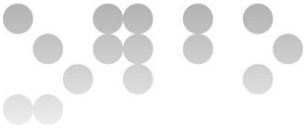


Dentro de la categoría de aplicaciones de mensajería síncronas tenemos las aplicaciones de chat de voz o de videollamada, donde ambas personas se comunican en tiempo real.

Para el caso de aplicaciones de mensajería asíncronas tenemos cualquier aplicación de correo electrónico o de chat. En estas aplicaciones, el emisor envía un mensaje que queda encolado en la bandeja de entrada del receptor, y este lo recibe en cuanto se conecte a la aplicación.

Por último**, el modelo más escalable es el de comunicación asíncrona**. Esto se debe a que la comunicación asíncrona hace que la mayor parte del volumen de trabajo recaiga sobre la petición realizada por el emisor, haciendo que la aplicación alojada en el servidor tenga menor carga y pueda realizar otras tareas sin necesidad de mantener la comunicación activa, y, por lo tanto, tenga un mejor rendimiento.

Este mejor rendimiento permite a la aplicación mantener un mayor volumen de trabajo, haciendo que esta sea más escalable.



Pregunta 4

Enunciado

Leer el siguiente artículo y responder a las preguntas:

Ray: A Distributed Framework for Emerging AI Applications

https://arxiv.org/pdf/1712.05889.pdf

a) Explica las principales ideas presentadas en este trabajo.

b) Describe la arquitectura distribuida, los modelos de comunicación y los mecanismos de tolerancia a fallos.

c) Proporciona tu propia perspectiva sobre el artículo e intenta encontrar las limitaciones del modelo propuesto.

Respuesta

a) Explica las principales ideas presentadas en este trabajo.

Inicialmente, en este trabajo se nos presentan las diversas funcionalidades de las aplicaciones que utilizan IA para interactuar con el entorno y aprender de él.

Se introduce brevemente la idea de cómo se ha llegado al punto tecnológico actual y al potencial futuro que presentan estas nuevas aplicaciones de IA, las cuales utilizan *reinforcement learning* (RL) para planear estrategias efectivas en base a la experimentación con datos.

En base a esto, se nos introduce el sistema distribuido Ray, un framework de computación en clúster de propósito general que permite, al mismo tiempo, la simulación, el entrenamiento y el servicio que necesitan estas nuevas aplicaciones de RL.

Al largo del artículo se nos presenta el funcionamiento, la arquitectura y las diferentes capacidades y respuestas de Ray ante los diversos problemas y requisitos que introducen estas nuevas aplicaciones de IA.

Igualmente, podemos ver el rendimiento de Ray ante los requisitos presentados (Latencia, escalabilidad, tolerancia a fallos, etc) mediante benchmarks y graficas de rendimiento, así como el planteamiento de evolución futura en respuesta al desarrollo de las propias aplicaciones.

b) Describe la arquitectura distribuida, los modelos de comunicación y los mecanismos de tolerancia a fallos.

La arquitectura de Ray está compuesta por 2 capas, una capa de aplicación y una capa de sistema.

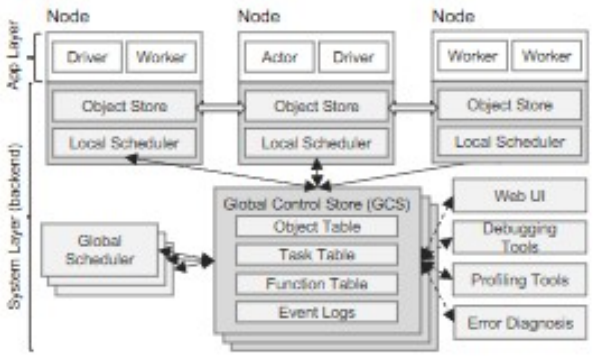


Ilustración 1 - Arquitectura de Ray

Como podemos ver en la imagen, Ray está distribuido en varios nodos controlados por un componente central.

Estos nodos presentan un *frontend* definido por la capa de aplicación y un *backend* que consiste en varios componentes de la capa de sistema, encargados de almacenar información y programar las tareas de forma local.

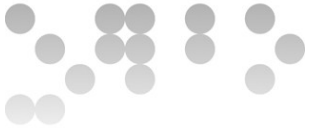
A su vez, el componente central es un *backend* compuesto por componentes de la capa de sistemas, que se encargan de gestionar y coordinar la información y programación de tareas de los nodos.

Vamos a ver los componentes de ambas capas y su función en el conjunto del sistema distribuido Ray.

La capa de aplicación se encarga de implementar la API y el modelo computacional, constando de 3 tipos de procesos: *Driver, Worker y Actor.*

Por otra parte, la capa de sistemas se encarga de implementar la programación de tareas y la gestión de datos, ambos con la finalidad de cumplir los requisitos de rendimiento y la tolerancia a fallos. La propia capa de sistemas consta de 3 componentes, cada uno de ellos escalable horizontalmente y con mecanismos de tolerancia a fallos: Un almacén de control global, o *global control store* (GCS), un programador de tareas distribuidas y un almacén de objetos distribuidos en la memoria.

Para hablar de los modelos de comunicación y los mecanismos de tolerancia a fallos hemos de entrar en profundidad en los componentes de la capa de sistema.



En primer lugar, el GCS encarga de mantener el control de todo el sistema, y es, esencialmente, un almacén de valores clave con funcionalidad Pub/Sub (Publicación y suscripción).

Esta funcionalidad Pub/Sub, característica por su alta disponibilidad, escalabilidad y latencia, consiste en un modelo de **comunicación asíncrona** donde un publicador (en este caso el GCS) envía mensajes catalogados en diferentes tipos de suscripciones a los propios suscriptores (En este caso, los nodos).

Mediante este tipo de centralización, donde toda la comunicación pasa por el GCS, Ray tiene su **mecanismo de tolerancia a fallos**, que consiste en que mantiene en el GCS la información referente a los objetos, tareas y funciones en varias tablas, y les envía dicha información a los nodos por separado, que a su vez la almacenan localmente. Esta información es la utilizada para escalar y fragmentar cada componente.

En el caso de que un nodo falle, solo debe recuperar la información correspondiente a su linaje del propio GCS. Esto, a su vez, permite que cada nodo escale de forma independiente su almacén de objetos y su programador de tareas.

En el caso del componente programador de tareas distribuidas, este sigue un modelo de **comunicación asíncrono y ascendente**, donde las tareas se transmiten desde los componentes más pequeños, los *drivers* y *workers* de la capa de aplicación, hasta el almacenamiento de tareas local de cada nodo, de allí al GCS y por último al almacén de tareas global, como podemos ver en la siguiente imagen:

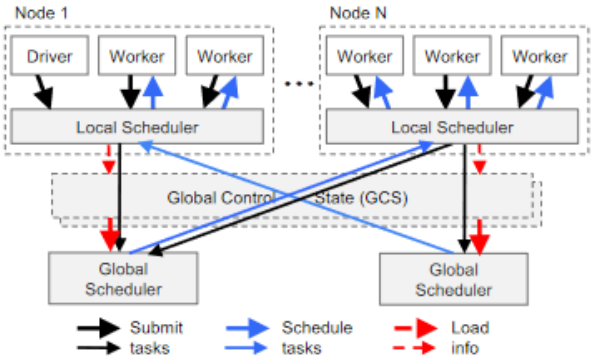
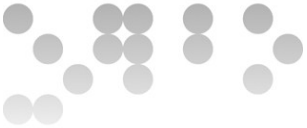


Ilustración 2 - Comunicación del componente programador de tareas

Si se produce algún cuello de botella se pueden instanciar nuevos programadores de tareas globales, utilizando la información almacenada en el GCS.

Por último, el almacén de objetos distribuidos en memoria, el cual almacena los *inputs* y *outputs* de cada tarea. Este modelo hace que *input* de tarea que llegue a un nodo desde el GCS, y no localmente, sea guardada en el almacén de objetos de dicho nodo. De la misma forma, todo *output* también es almacenado en dicho almacén.



Toda esta información, y su traslado desde y hasta el GCS se lleva a cabo con una **comunicación asíncrona**, como hemos visto en la descripción del GCS.

De esta forma, todos los objetos son tratados de forma independiente, y no son actualizables, ya que se trabaja con un modelo de datos inmutable. Que los objetos no sean actualizables permite al GCS tener un control de los objetos almacenados en cada nodo, y permite recuperar la información de estos en caso de que haya un error, de nuevo, como parte del sistema de **mecanismos de tolerancia a fallos** de Ray.

Con esto podemos concluir que Ray utiliza un modelo de **comunicación asíncrono**, basado en la transmisión de datos desde sus diversos componentes, y centralizadas en los almacenes de datos del GLS.

Esta misma centralización permite que haya **mecanismos de tolerancia a fallos,** uno basado en el linaje para las tareas y los actores, y otra basada en la replicación para el almacén de metadatos.

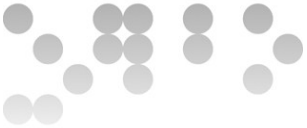
c) Proporciona tu propia perspectiva sobre el artículo e intenta encontrar las limitaciones del modelo propuesto.

En mi opinión, Ray supone un avance en el desarrollo de sistemas distribuidos, ya que podemos ver que desde un sistema de almacenamiento y gestión centralizado es posible el escalar enormemente el sistema distribuido, adaptándolo así a las, cada vez mayores, necesidades de las nuevas aplicaciones de IA que se nos plantean en el artículo.

Como limitaciones puedo señalar, principalmente, a que Ray solo cumple las características necesarias para el soportar el servicio, el entrenamiento y la simulación en el contexto de las aplicaciones de RL. Como se nos indica en el texto, para otros contextos hay aplicaciones que cumplen mejor dicha función.

Personalmente, veo una limitación el hecho de que todo el mecanismo de tolerancia a fallos este centralizado en un modelo de almacenamiento central de información, focalizado en el GLS.

Esta centralización puede provocar que, si este mismo componente tiene algún problema, como corrupción de datos, se trasmita a todos los nodos del sistema. Igualmente, si este componente deja de estar operativo, el propio sistema se vendría abajo.



Pregunta 5

Enunciado

Mira el siguiente video:

How we've scaled Dropbox

https://www.youtube.com/watch?v=PE4gwstWhmc

a) Describe las principales ideas presentadas en este video.

b) Describe la arquitectura distribuida, los modelos de comunicación y las técnicas de escalado empleadas.

c) Proporciona tu propia perspectiva sobre la arquitectura de Dropbox e intenta encontrar las limitaciones de este modelo. ¿Es factible en 10 años con redes 6G?

Respuesta

a) Describe las principales ideas presentadas en este video.

En el video se nos presenta la evolución de la arquitectura y el backend de la aplicación Dropbox para adaptarse a los desafíos tecnológicos que encontraron sus desarrolladores.

Nos trasmiten la idea de que el desarrollo de Dropbox fue especialmente difícil, debido a que no podían aprovecharse las soluciones habituales utilizadas por otros aplicativos para hacer frente a los 2 principales problemas que encuentran los sistemas distribuidos hoy en día.

La primera es un alto volumen de escritura de datos, que, a diferencia de otras aplicaciones como Twitter, en Dropbox es el mismo que el de lectura.

La segunda consiste en los requisitos de calidad y de medidas para controlar los errores. En Dropbox son muy altos, ya que la aplicación es un gestor de archivos en la nube, y, por lo tanto, no pueden permitirse errores donde, por ejemplo, se publiquen o eliminen archivos de forma incorrecta.

En conjunto, se trasmite la idea de cómo es trabajar en una *startup* desde sus inicios, con un proyecto que crece constantemente y a una gran velocidad, al igual que los propios recursos disponibles para hacer frente a dicha demanda, y de cómo han hecho frente tecnológicamente a las diversas dificultades que han encontrado, mostrando paso a paso la evolución de su arquitectura.

b) Describe la arquitectura distribuida, los modelos de comunicación y las técnicas de escalado empleadas.

En el video se nos muestra la evolución de la arquitectura de Dropbox desde 2007, donde era solo un servidor, hasta el 2012, fecha del video. Podemos ver dicha arquitectura final de 2012 en la siguiente imagen:

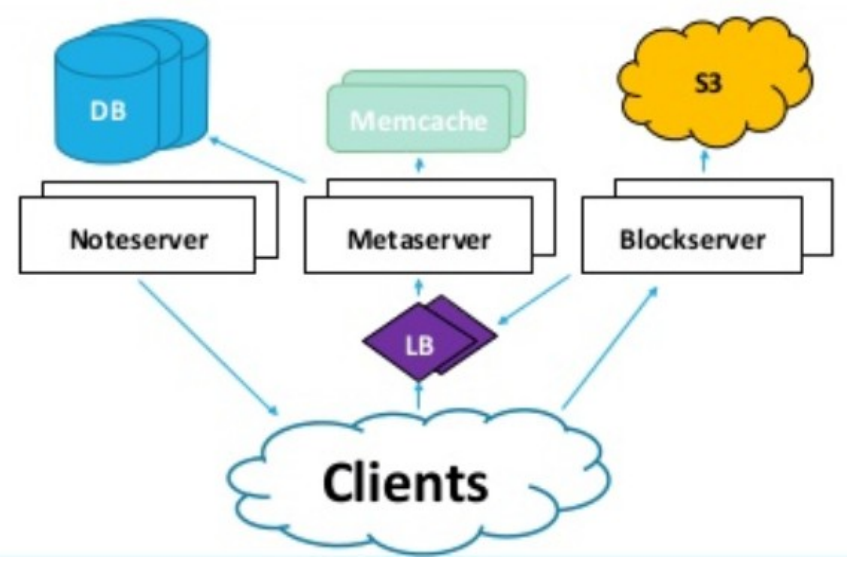


Ilustración 3 - Arquitectura de Dropbox, obtenida de slideshare (Dropbox - Architecture and Business Prospective)

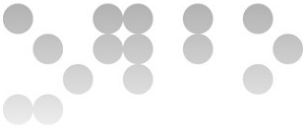
Podemos ver los siguientes componentes:

- Load Balancing (LB), se trata de un balanceador de carga, utilizado para optimizar recursos, minimizar el tiempo de respuesta y evitar la sobrecarga de datos.
- Noteserver, se trata de un servidor encargado de enviar notificaciones pushup a los clientes.
- Metaserver, hace todas las llamadas relacionadas con los metadatos de los archivos.
- Blockserver, se encarga del hosting del contenido de los archivos.
- Data Base (DB), base de datos relacional basada en MySQL, donde se almacenan todos los metadatos de archivos de los usuarios.
- Memcache, se trata de un servicio de la App Engine de Google que guarda en memoria la cache de las consultas, con la finalidad de acelerar las consultas a la BD y aligerar la carga de trabajo sobre la misma.
- Amazon Simple Storage (S3), almacén de datos en la nube donde se almacenan los archivos de los usuarios.

Todos los componentes de la arquitectura distribuida, exceptuando el servicio de almacenamiento externo de Amazon S3, son **escalables horizontalmente**, y pueden añadirse mas de los mismos según aumenta la demanda del servicio.

Este escalado horizontal es el que se ha ido utilizando a lo largo de la evolución del proyecto, ampliando el número de cada componente según aumenta el consumo del servicio.

El funcionamiento de la arquitectura es simple, los clientes realizan **comunicaciones asíncronas**, solicitando subidas o bajadas de archivos. Estas comunicaciones se dirigen a la vez



tanto hacia el Blockserver como hacia el balanceador de carga (LB) . El Blockserver se comunica con el balanceador de carga mediante RPC.

El hilo de comunicación del Blockserver recuperara los archivos del servidor S3, y, posteriormente, contactara con el balanceador de carga para transmitir los metadatos de dichos archivos.

El hilo de comunicación del balanceador de carga contacta con el metaserver para transferir la información de los metadatos de los archivos relacionados con la transacción, esta comunicación puede llegar tanto desde el cliente como del Blockserver.

El Metaserver realizara las transacciones de metadatos, almacenándolas o recuperándolas de la base de datos, según el tipo de requerimiento del usuario. En este momento interviene el Memcache, donde se cachearán y reutilizarán estas transacciones si fuera posible, con el objetivo de reducir la carga de la base de datos.

Por último, el Metaserver se comunicará con el Noteserver para que este envié una notificación al cliente, informándole de la finalización de la transacción.

Como método de tolerancia a fallos y disponibilidad se usa la replicación.

c) Proporciona tu propia perspectiva sobre la arquitectura de Dropbox e intenta encontrar las limitaciones de este modelo. ¿Es factible en 10 años con redes 6G?

La arquitectura de Dropbox, aunque correcta, es el fruto de una ampliación de una arquitectura básica sobre la marcha, según ha ido aumentando el volumen y las necesidades del proyecto.

Una vez se han incluido en la misma todos los componentes requeridos para su correcto funcionamiento y optimización, como pueden ser los balanceadores y el Memcache, ambos utilizados para optimizar el flujo de datos, parece que desde Dropbox se han limitado a ampliar su arquitectura utilizando un **escalado horizontal**.

Aunque este modelo de ampliación les ha funcionado hasta el año del video, según aumenten exponencialmente el numero de usuarios y, al mismo tiempo, las tecnologías evolucionen (Como el aumento de velocidad de navegación introducidos por el 4G y el 5G), el modelo podría volverse insostenible, debido al aumento exponencial de componentes que deberían replicar para seguir manteniendo viable el sistema.

Respecto a si el modelo sería factible con redes 6G, podría serlo, pero, con la finalidad de que la aplicación siga prestando un servicio acorde a los estándares de Dropbox hasta la fecha, el número de cada componente habría de ser incrementado de manera exponencial, lo cual podría ser muy costoso de mantener y, por lo tanto, dejar de hacer a la aplicación económicamente viable.

Bibliografía

- COLOURIS, GEORGE (2012). Distributed Systems, Comcepts and Design, Fifth Edition.Boston: Addison-Wesley

- OpenWebinars, Articulo: Qué es un sistema distribuido y qué ventajas aporta su funcionamiento: https://openwebinars.net/blog/que-es-un-sistema-distribuido/

- Marco de desarrollo de la Junta de Andalucía: Conceptos sobre la escalabilidad: http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/220

- Blog de Oscar Blancarte, Articulo: Escalabilidad Horizontal y Vertical: https://www.oscarblancarteblog.com/2017/03/07/escalabilidad-horizontal-y-vertical/

- A Fresh Graduate's Guide to Software Development Tools and Technologies, Chapter 6: Scalability: https://www.comp.nus.edu.sg/~seer/book/2e/Ch06.%20Scalability.pdf

- Blog P&A, Articulo: Comunicación sincrónica y asincrónica: conceptos y herramientas: https://blog.grupo-pya.com/comunicacion-sincronica-asincronica-conceptos-herramientas/

- TechTarget, Articulo: Synchronous vs. asynchronous communications: The differences: https://searchapparchitecture.techtarget.com/tip/Synchronous-vs-asynchronous-communication-The-differences

- IIC, Articulo: Aprendizaje por refuerzo & Optimización: https://www.iic.uam.es/inteligencia-artificial/aprendizaje-por-refuerzo/

- Google Cloud: guía arquitectura Cloud Pub/Sub: https://cloud.google.com/pubsub/architecture?hl=es-419

- Slideshare: Dropbox - Architecture and Business Prospective**:** https://es.slideshare.net/ChiaraCilardo/dropbox-architecture-and-business-prospective

- Dropbox - Entre bastidores: descripción de la arquitectura**:** https://www.dropbox.com/es_ES/business/trust/security/architecture