1. Explica el concepto de escalabilidad con ejemplos en sistemas distribuidos Web. ¿Cómo escalarías una aplicación Web si piensas que recibirá un "slashdot effect"?

Google sería un gran ejemplo de un sistema distribuido web. Si nos centramos en su modelo de negocio original, es un buscador que se basa en las palabras clave que introduce un usuario. Aprovecha la interactividad con ese usuario para mostrar publicidad asociada o bien en los resultados del buscador o bien en los diferentes soportes (páginas web) en las que aterriza.

Los sistemas distribuidos están formados por una red conectada con varios ordenadores que comparten recursos. Estas redes pueden estar distribuidas geográficamente y en el caso de Google, se trata de una red distribuida a nivel mundial que sirve las peticiones enrutando las conexiones más cercanas al usuario.

El proyecto original se llamó BackRub y nació en 1998 como un proyecto universitario. Su modelo es un gran ejemplo de un sistema distribuido con varios subsistemas que ofrece servicio a millones de usuarios. En 2009 tenía unos 45.000 servidores tan solo en uno de sus centros de datos. Hoy en día sus servidores han crecido de forma exponencial y es un ejemplo de arquitectura 100% escalable.

En la red de Google cada una de las máquinas que forma la red, funciona de forma independiente, pero con el objetivo común de dar respuesta a una pregunta del usuario. De esta forma, el usuario visualiza un número de páginas servidas (resultado de búsqueda) en una experiencia unificada.

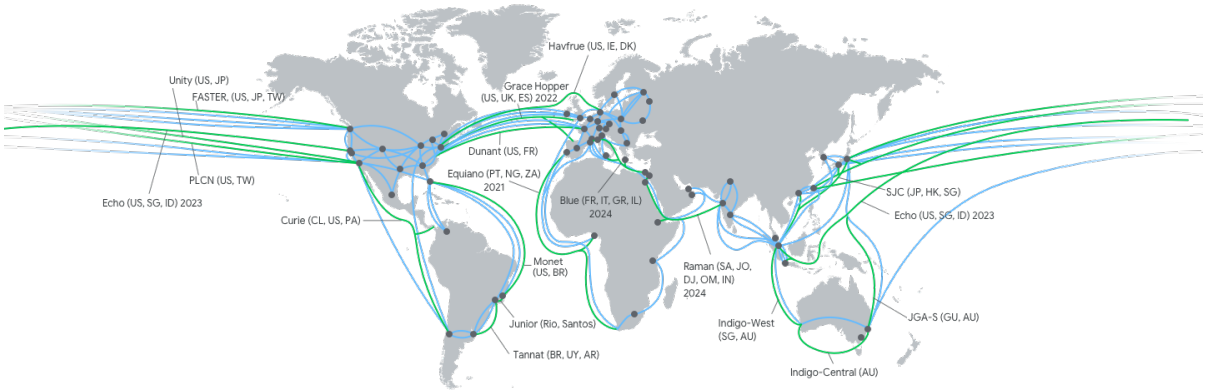
Cada conexión es redirigida a través de balanceadores de carga que distribuyen las peticiones a los diferentes servidores para garantizar el servicio.

La aparición de la nube hace que los sistemas de hoy en día sean aún más escalables y ofrecen servicios más sencillos de gestionar (serverless) donde ni siquiera es necesario comprar una máquina e instalarla, tan sólo hay que decidir el tipo de ampliación que se necesita en función de cuáles sean las necesidades. Si, como en el caso de Google, se necesita atender un mayor número de conexiones, la nube te permite escalar tanto verticalmente como horizontalmente.

El escalado vertical consiste en aumentar los recursos de una sola máquina para ofrecer una mayor capacidad de proceso (memoria, CPU, almacenamiento y elementos de red) mientras que el escalado horizontal consiste en ampliar el número de nodos a la red (cada uno con sus propios recursos). Los nodos atenderán peticiones de forma concurrente.

Para que esos nodos puedan funcionar correctamente y no se saturen por un exceso de peticiones, se coloca un balanceador de carga que distribuye las conexiones en función de la capacidad de cada nodo. Para el usuario es transparente la forma en la que se amplía la capacidad de la red.

Ambas formas de escalabilidad no son excluyentes entre sí. En el caso de Google, es un gran ejemplo de escalabilidad horizontal, su red ha ido creciendo de forma horizontal a medida que era necesario incrementar el número de peticiones de usuarios llegando a tener un sistema en la nube que abarca toda la geografía mundial.



Fuente: https://cloud.google.com/about/locations#network

También Google ha usado la escalabilidad vertical en su red, especialmente cuando ha sido necesario realizar cambios tecnológicos porque los dispositivos quedan obsoletos y en esos casos, la modificación de un servidor lleva implícita una ampliación (nueva tipología de memoria, discos SSD, etc.).

Actualmente Google ofrece servicios de Cloud (similares a los que ofrece Microsoft Azure o Amazon Web Services) a sus clientes y ofrece un modelo de escalado vertical y horizontal. Este sería un ejemplo de su servicio estándar de escalabilidad vertical:

E2 estándar | E2 con alta capacidad d... | CPU elevada E2 | Núcleo compartido E2

Los tipos de máquinas estándar E2 tienen 4 GB de memoria de sistema por CPU virtual.

Tipos de máquina	CPU virtuales*	Memoria (GB)	Cantidad máxima de discos persistentes (PD)†	Tamaño total máximo de PD (TB)	SSD local	Ancho de banda de salida máximo (Gbps)‡
e2-standard-2	2	8	128	257	No	4
e2-standard-4	4	16	128	257	No	8
e2-standard-8	8	32	128	257	No	16
e2-standard-16	16	64	128	257	No	16
e2-standard-32	32	128	128	257	No	16

Fuente: https://cloud.google.com/compute/docs/general-purpose-machines

Escalabilidad para el caso slashdot

Si disponemos de una aplicación web, por ejemplo, una página web de una campaña promocional de marketing, a priori requiere alta disponibilidad y capacidad para atender un número estimado de conexiones. Normalmente, las campañas de marketing se estiman en base a unos parámetros iniciales de audiencia y ratios de conversión (audiencia estimada, ratio de conversión a visita, lead y captación).

Las necesidades del sistema distribuido se estiman en base a esas previsiones iniciales, pero debe tener capacidad suficiente para poder crecer y atender a un número mayor de peticiones en caso de

que se produzca un pico de actividad durante la campaña. En este caso, se recomienda una escalabilidad vertical para campañas de duración corta, de nicho y estacionales.

En este tipo de campañas, se dispone de un caso de éxito previo y se conoce la previsión de datos que permiten el cálculo inicial del dimensionamiento de la máquina y la tasa esperada de crecimiento vertical. El cambio es rápido y no requiere instalación o distribución de software en distintas máquinas. También es más económico. El riesgo que entraña esta solución es que no aporta mucho beneficio en alta disponibilidad y está limitado a la capacidad de ese servidor.

Sin embargo, si la campaña promocional de marketing es muy novedosa, disruptiva y tiene una duración indeterminada, que depende del éxito de la campaña, puede **sufrir un efecto slashdot**, es decir, que haya un incremento masivo y repentino para el que hay que estar preparado. Esta decisión es más estratégica, permite un balanceo de carga de las peticiones y además no tiene límites. Sin embargo, requiere de una mayor inversión y previsión, ya que es necesario que la aplicación web esté construida de forma correcta para permitir ese tipo de escalabilidad y tiene una mayor complejidad, puesto que hay que instalar y distribuir el software en cada máquina.

Hoy en día esta complejidad está reduciéndose debido a las mejoras que aportan los contenedores (docker o kubernetes).

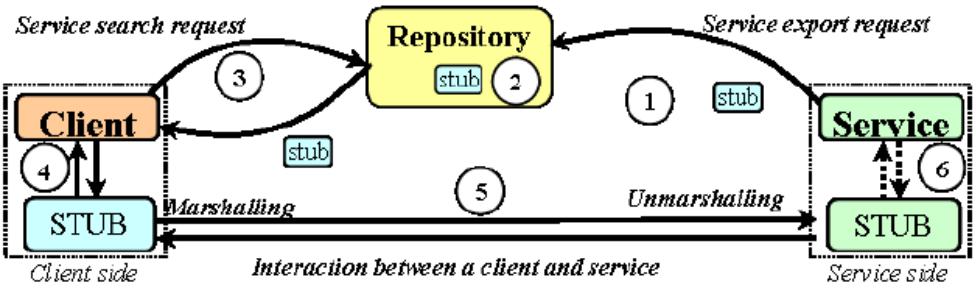
Fuentes utilizadas:

- https://en.wikipedia.org/wiki/Slashdot_effect
- https://cloud.google.com/about/locations#network
- https://cloud.google.com/compute/docs/general-purpose-machines
- https://www.universidadviu.com/es/actualidad/nuestros-expertos/sistemas-distribuidos-caracteristicas-y-clasificacion
- https://www.arsys.es/blog/soluciones/escalado-horizontal-vs-vertical
- https://sistemasdistribuidos.foroactivo.com/t137-google-un-sistema-distribuido-de-talla-internacional-o http://vis.usal.es/rodrigo/documentos/sisdis/teoria/9-google.pdf
- https://azure.microsoft.com/en-gb/topic/kubernetes-vs-docker/

2. Compara el middleware RPC con el publish/subscribe con ejemplos de uso.

RPC (Remote Procedure Call) permite que una aplicación pueda hacer uso de procedimientos que están en una aplicación remota como si esos procedimientos estuvieran en la propia aplicación local. Encapsula funcionalidades a través de esas llamadas remotas, no es necesario conocer el detalle cómo está implementado ese procedimiento, sólo hay que conocer qué hace, cómo se tiene que hacer una llamada formal y qué parámetros deben enviarse para poder recibir una respuesta.

En el siguiente diagrama se puede observar a través del ejemplo de Enterprise Java Bean (Interfaz remota). Para poder entender cómo funciona es necesario conocer algunos conceptos:



Fuente : https://www.researchgate.net/figure/RPC-based-middleware-architecture_fig8_226623184

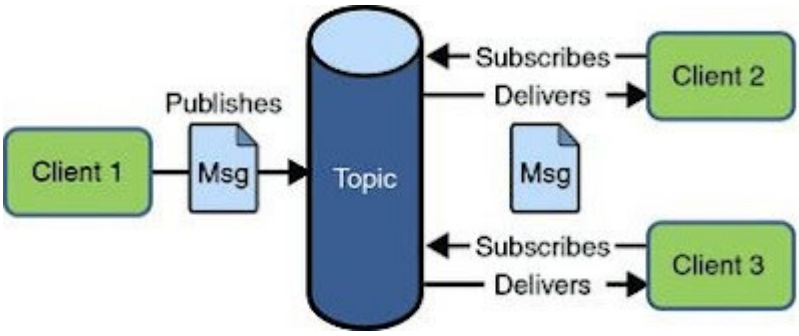
- STUB: es la interfaz donde se definen los parámetros de entrada y salida que necesita la llamada al procedimiento.
- Repository: es donde se publican los servicios a los que va a poder acceder un cliente.

Una vez se ha definido la interfaz y se ha publicado en el repositorio de servicios, el cliente recupera el stub y genera el código necesario para usar el servicio.

Cuando realiza una llamada envía los parámetros en la codificación conocida por ambas partes y el servidor recoge esos parámetros, ejecuta el procedimiento/función y devuelve el resultado esperado. Esta comunicación se produce de forma síncrona, por lo tanto, el cliente conoce al servidor y éste a su vez, puede tener muchos clientes que le conocen.

Un ejemplo de aplicación práctica sería la consulta de la temperatura o de un servicio de meteorología en tiempo real, por ejemplo, la consulta que hacemos del tiempo en nuestro móvil (servicio de meteorología de Google) cuando de forma activa queremos saber el tiempo en ese momento, también se podría implementar con RPC para que nos diera la información actualizada.

Publish/subscribe es un servicio de mensajería donde los clientes publican o se suscriben a colas de mensajes. Se producen intercambios de mensajes entre distintas aplicaciones a través de un cluster centralizado. Un ejemplo sería el Java Messaging Service en modo publicación/suscripción:



Fuente: https://sites.google.com/site/sureshdevang/java-message-service-jms

- Topic: es un descriptor de la cola de mensajes donde se van a publicar

Un cliente publica un mensaje en este topic (asunto) y distintos clientes recuperan estos mensajes porque se han suscrito a esa cola.

Una aplicación práctica de este tipo de servicio es un grupo de WhatsApp, un miembro del grupo envía un mensaje al grupo y todos los miembros del grupo reciben ese mensaje (están "suscritos" a la cola de mensajes de ese grupo) y a la vez cualquiera de ellos puede publicar. Volviendo al mismo ejemplo anterior sobre la consulta del tiempo en el móvil, el servicio de Google mantiene información actualizada, aunque nosotros no estemos preguntando por el tiempo de forma activa, en

este caso está suscrito a un servicio de mensajería que le está respondiendo cuando tiene datos actualizados.

Este tipo de servicio es asíncrono y en este caso, ninguno de los publicadores/suscriptores tiene que conocerse entre sí para intercambiar mensajes.

Como se puede ver el middleware RPC y el middleware de mensajería (MOM) no comparten muchas características comunes, ni en uso ni en funcionalidad. Ambos están basados en el mismo principio, es decir es un software que permita la comunicación y conexión entre diferentes aplicaciones o sistemas en una red distribuida, En resumen, sus diferencias principales son:

- Forma de comunicación: mientras que el RPC es síncrono, MOM es asíncrono
- Formato de mensaje: en RPC se intercambian parámetros y en MOM datos.
- Propósito: en RPC es una llamada a un procedimiento que proporciona una respuesta a una serie de parámetros (lógica de negocio) y en MOM se intercambia información (mensajes) entre sistemas.

Fuentes:

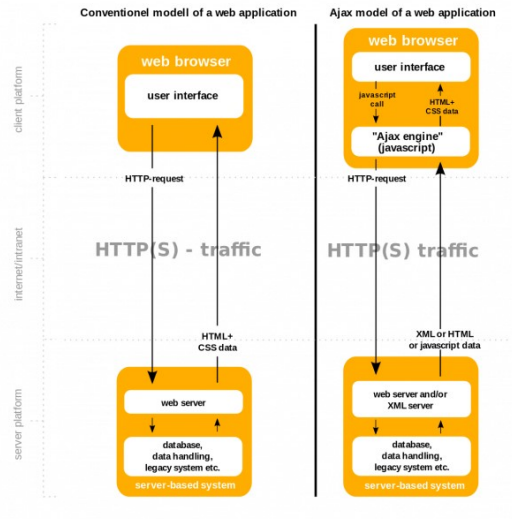
- https://es.wikipedia.org/wiki/Enterprise_JavaBeans
- https://es.wikipedia.org/wiki/Llamada_a_procedimiento_remoto
- https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern
- https://sites.google.com/site/sureshdevang/java-message-service-jms
- https://cs.gmu.edu/~menasce/papers/IEEE-IC-MOM-RPC-MarchApril2005.pdf
- https://programmerclick.com/article/73341368514/
- https://ieeexplore.ieee.org/document/1405980
- https://www.researchgate.net/figure/RPC-based-middleware-architecture_fig8_226623184
- https://www.connecting-software.com/es/blog/what-is-middleware-after-all/
- https://www.ibm.com/cloud/learn/middleware

3. Compara los modelos de comunicación asíncrono vs síncrono en aplicaciones web (AJAX). Pon ejemplos de cuándo utilizar uno u otro.

El intercambio de información de las aplicaciones web a través de Internet se puede realizar de forma síncrona o asíncrona.

La comunicación entre el emisor y el receptor es independiente tanto de la distancia a la que se encuentren como del lugar. Para que esta comunicación se puede producir de forma inmediata o al menos en un período de tiempo razonable, tiene que existir inmediatez y fluidez en la comunicación para que se pueda considerar una comunicación síncrona, sin embargo, en la comunicación asíncrona existe una diferencia significativa entre el envío del mensaje y la recepción de este.

La comunicación síncrona tiene algunas limitaciones, entre ellas se encuentra la actualización de los datos. Por ejemplo, si tenemos una aplicación web que está solicitando datos de las ventas online cada X tiempo (por ejemplo, cada 5 segundos), siendo síncrono hay un pequeño espacio de tiempo en el que los datos van a estar desactualizados. El usuario tiene que esperar a que el servidor obtenga los datos y los envíe y vuelva a cargar la página para que se puedan ver los resultados. La experiencia de usuario por lo tanto es peor.



Fuente: https://www.hostinger.es/tutoriales/que-es-ajax

Para resolver este problema se ha desarrollado la comunicación asíncrona. AJAX (Asynchronous Javascript and XML) es un conjunto de técnicas de desarrollo web que procesa las peticiones de los servidores en segundo plano. Se resuelven este tipo de problemas de desactualización de datos, porque la aplicación web puede recuperar los datos del servidor sin tener que volver a cargar toda la página. (en este caso también AJAX puede funcionar de modo síncrono sin tener que recargar toda la página).

Realmente el modo asíncrono tiene una respuesta más rápida y no interrumpe la fluidez en la experiencia de usuario. También es un método más eficaz, porque las peticiones se hacen bajo demanda y no programadas como las llamadas síncronas. La comunicación síncrona se recomienda cuando hay que ejecutar una tarea en concreto y es la única que está activa y requiere que un emisor y receptor "hablen" simultáneamente.

El mejor ejemplo que existe hoy en día de comunicación síncrona son los chats y las videollamadas (WhatsApp, Telegram, Messenger, Dúo). El usuario quiere comunicar con otras personas y pueden compartir momentos en tiempo real.

Por otro lado, se recomienda la comunicación asíncrona cuando la aplicación web tiene un entorno multitarea, donde pueden convivir tareas en ejecución y en segundo plano simultáneamente. Entre los mejores ejemplos de comunicación asíncrona se encuentran las notificaciones de tendencias de Twitter, las historias que se muestran en las redes sociales, salas de chat, informes bursátiles, etc. El usuario está usando la aplicación en su funcionalidad principal y la aplicación le va informando de otras funciones adicionales que puede consultar cuando desee.

AJAX asíncrono siempre será más recomendado para funciones que no requieren inmediatez y requieren un nivel de atención e interacción con el usuario bajo. De la misma forma AJAX en su funcionalidad síncrona, es el más recomendado para aplicaciones donde sólo se realiza una tarea y la información se desea tener casi "en tiempo real".

Fuentes:

- https://www.hostinger.es/tutoriales/que-es-ajax
- https://azsalud.com/cultura-y-reflexion/comunicacion-sincrona-asincrona
- https://programmerclick.com/article/3560593432/
- http://www.sergiomadrigal.com/2014/03/24/aplicaciones-sincronas-y-asincronas/

- https://es.wikipedia.org/wiki/Comunicaci%C3%B3n_sincr%C3%B3nica
- https://blog.weremoto.com/2020/10/15/comunicacion-sincronica-vs-asincronica-cuando-conviene-utilizar-cada-una/
- https://asana.com/es/resources/synchronous-vs-asynchronous-communication
- https://mx.indeed.com/orientacion-profesional/desarrollo-profesional/diferencias-comunicacion-sincronica-asincronica

4. Bitcoin: A Peer-to-Peer Electronic Cash System

a) Explica las principales ideas presentadas en este trabajo.

El informe de Satoshi Nakamoto presenta la idea de cómo generar dinero virtual usando una arquitectura distribuida de nodos punto a punto eliminando intermediarios y autoridades centrales que controlen y verifiquen todo el proceso.

Propone un cambio en el paradigma financiero, cambiando el dinero fiat (basado en la confianza) por la certeza matemática, donde las transacciones son verificadas y validadas criptográficamente y son datos que se pueden trazar y consultar de forma pública, aunque anónima, sin que sea necesario que haya una entidad validadora que lo certifique. El propio proceso es el certificador.

Introduce el concepto colaborativo en la capacidad de procesamiento. Hasta ahora las soluciones tecnológicas requerían de una entidad central que controla la generación de dinero. Las diferentes entidades se comunican entre sí, establecen acuerdos, intercambio de divisas y añaden una cadena de intermediarios que suman costes a cada transacción monetaria. Detrás de este sistema financiero hay una inversión en infraestructura muy potente para dar cobertura a todo el sistema.

Blockchain ofrece la posibilidad de eliminar estos intermediarios sin tener que realizar una inversión en una gran infraestructura, porque ésta se distribuye entre nodos que participan de forma voluntaria en la red, aportando la capacidad de procesamiento. La red ofrece incentivos que premian a los nodos por su aportación en la generación de bloques, bien mediante la creación de nueva moneda (mineros) o bien mediante comisiones por procesamiento de bloques de transacciones.

La idea principal es que la tecnología Blockchain es lo suficientemente segura como para tener una aplicación práctica en uno de los puntos más delicados de toda sociedad: el dinero. En definitiva, sustituye la necesidad de confiar en entidades por tecnología.

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

La arquitectura que propone Satoshi Nakamoto es una arquitectura distribuida donde los nodos se conectan entre sí punto a punto. Esta solución está formada principalmente **Servidores Timestamp distribuidos** que están conectados a la red en un momento determinado (nodos).

Servidor Timestamp

El servidor es el que agrupa transacciones en un bloque y las marca con un hash. Cada bloque tiene su propio hash (función que encripta y genera salidas sobre una información de entrada proporcionada). Esta marca o hash es una prueba de que los datos son reales y asegura que el bloque existe porque éste se inserta en el propio hash.

Cada bloque de transacciones contiene el hash del bloque anterior y el del siguiente, formando una cadena de bloques que los unen entre sí.

Uno de los elementos más importantes en la implementación del servidor Timestamp distribuido es la generación de las "pruebas de trabajo" (proof of work).

Forma parte de la verificación de un bloque de transacciones, cada vez que un bloque se distribuye en la red, se realiza una validación del hash (SHA256). La validez del hash depende de que exista un número determinado de ceros al inicio de propio hash. El cálculo se realiza con la combinación del hash del bloque anterior más un número que sólo se utiliza una única vez. El resultado de la combinación de ambos números, cumpliendo la condición mencionada (número determinado de ceros al inicio del hash), es el identificador del siguiente bloque. Ese número determinado de ceros dependerá del número de bloques que se generan en una determinada unidad de tiempo. Si se generan demasiados bloques se aumenta la dificultad.

Una vez que la prueba de trabajo se ha generado, el bloque no se puede modificar. Si alguien intentase modificar de alguna forma la cadena, necesitaría generar una cadena más larga de la que existe en ese momento. Para eso debería tener mayor capacidad de cálculo que todos los nodos que forman la red distribuida de Blockchain.

Cada nodo verifica la validez de la cadena y se alcanza un consenso por mayoría. La mayoría no viene determinada por dirección IP (una IP un voto), porque entonces alguien podría acumular muchas IPs y alterar la votación. La propia prueba de trabajo es un voto (CPU). La cadena más larga es la validada porque es la que tiene una mayor inversión en tiempo de proceso.

Cuanto mayor sea la capacidad de procesamiento de los nodos, más grande se hará la cadena y más difícil será atacar el sistema, porque necesitaría modificar todos los bloques anteriores de la cadena y volver a ponerse al día con todos los bloques que se hubieran generado posteriormente. Es como tratar de rellenar huecos en un libro contable en el que se han borrado transacciones, es fácilmente detectable.

Modelo de comunicación

La red está formada por un conjunto de nodos conectados punto a punto donde se realizan las siguientes tareas:

- Transmite las transacciones a los nodos
- Cada nodo recopila las nuevas transacciones en un bloque
- Cada nodo procesa la prueba de trabajo para su bloque
- Transmite la prueba de trabajo procesada a todos los nodos
- Los nodos de la red validan el bloque sólo si las transacciones son válidas y aún no se han gastado
- Los nodos aceptan el bloque trabajando en uno nuevo donde el hash se basa en el hash del bloque validado.

En la red se están generando continuamente nuevos bloques, por lo que hay diferentes versiones de la cadena distribuyéndose, la cadena más larga será siempre la que prevalece porque, si un nodo estaba validando un bloque que era "el último de la cadena" y aparece otra cadena que tiene ese bloque como bloque intermedio, implica que ya se ha validado previamente. Todos los nodos que hubieran estado trabajando en la validación de ese bloque lo darán por válido cuando encuentran una cadena más larga en la que ese bloque no es el "último" o no esté incluido.

Los nodos a su vez, por ese procesamiento que aportan a la red, se pueden llevar incentivos o bien por la generación de nuevas monedas (mineros) o bien mediante comisiones por la generación de bloques.

Como se trata de un bien "virtual" el gasto que se produce en cada nodo está asociado al consumo de electricidad y tiempo de CPU que pone al servicio de la red.

Tolerancia a los fallos

La red es muy robusta por su sencillez, los nodos se conectan entre sí y requieren propia coordinación. Entran y salen de la red cuando desean, pueden validar una prueba de trabajo que se ha producido cuando no estaban, votar (CPU) etc. El sistema no pierde transacciones ni bloques porque está preparado para solucionar este tipo de problemas.

Las nuevas transacciones no tienen que llegar a todos los nodos, pero con el tiempo llegan a un número suficiente de nodos para que se incluyan en un bloque. La difusión de bloques también es tolerante a fallos, porque si un nodo no recibe un bloque lo solicitará cuando reciba el bloque siguiente y se de cuenta de que no ha procesado ese bloque.

Se puede dar el caso de que dos nodos transmitan versiones diferentes del bloque siguiente al mismo tiempo, en este caso se generarían dos versiones de la cadena con la misma longitud de bloques y cuando se reciba la siguiente prueba de trabajo y la cadena sea más larga, se valida la cadena más larga.

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

El artículo presenta una aplicación práctica de una tecnología que lleva presente en el mercado desde hace mucho tiempo. El hecho de que esta aplicación gire en torno al sistema financiero añade más controversia a su uso.

Uno de sus puntos más fuertes es su protección y su poca vulnerabilidad frente a ataques por la naturaleza criptográfica de su moneda y la distribución descentralizada de la red que conforma el sistema.

Esta tecnología avala un sistema de transacciones financieras distribuida que no necesita ser vigilado. Es precisamente esa libertad la que ha puesto a la comunidad financiera en contra de este sistema alternativo, aunque muchos bancos están implementando ya Blockchain y se augura un futuro digital para las monedas de algunos países como respuesta "contra" el sistema actual de Bitcoin, aunque no use Blockchain como base tecnológica.

Sin embargo, a pesar de las reticencias del sector financiero, algunos países están aceptando ya Bitcoin como moneda válida, tratando de regularizar la circulación de este dinero virtual que está siendo cada vez más potente como mercado alternativo.

La privacidad que proporciona Blockchain es también uno de sus puntos de mayor controversia. En el sistema financiero actual se garantiza un nivel mínimo de privacidad, que, por otro lado, también permite la existencia de "paraísos fiscales". Con el uso de Blockchain, aunque la información es pública y trazable también es anónima.

La limitación principal que tiene este sistema es el uso de CPU, que crece de forma exponencial a medida que va creciendo la cadena de bloques. Exige un consumo energético que no se considera sostenible ni bueno para el medio ambiente. Este es un de los puntos que más ha influido negativamente en la visión que tiene el público generalista sobre este sistema.

Otra de las limitaciones es la capacidad de procesamiento. Mientras que la mayoría de los nodos que forman la red de Blockchain tengan mayor capacidad de procesamiento que el conjunto de nodos que podrían unirse para atacarla, el sistema es seguro. ¿Cómo se garantiza entonces la fiabilidad de cada

nodo? ¿Cómo podemos asegurar la honestidad de los nodos que participan en la red? No hay forma de saberlo y por eso, a pesar de que la tecnología garantiza la fiabilidad de las operaciones, no es posible garantizar este escenario.

El futuro de Blockchain tiene sin embargo muchas aplicaciones prácticas fuera del ámbito monetario. También puede aplicarse en la generación de bienes digitales (NFT) o la gestión de contratos inteligentes (eliminando intermediarios como notarías, organismos públicos etc.).

También puede ser un mecanismo para garantizar la trazabilidad de cualquier bien para controlar la cadena de suministro. En Europa existe un proyecto en este sentido orientado a la certificación de la producción agrícola, en un intento de ofrecer transparencia en la distribución de las materias primas desde su lugar de origen.

Fuentes:

- https://finanzasparamortales.es/que-es-el-dinero-fiat/
- https://www.santander.com/es/stories/blockchain-seguridad-y-transparencia-al-servicio-de-la-banca
- https://www.bbva.com/es/economia/mercados-financieros/blockchain/
- https://computerhoy.com/noticias/tecnologia/hay-paises-adoptando-bitcoin-como-moneda-oficial-sera-espana-algun-momento-929947
- https://www.elblogsalmon.com/conceptos-de-economia/que-euro-digital-que-se-diferencia-bitcoin-otras-criptomonedas
- https://www.eleconomista.es/mercados-cotizaciones/noticias/11242451/05/21/La-huella-que-deja-el-Bitcoin-en-el-medio-ambiente-amenaza-el-futuro-de-la-divisa.html
- https://es.wikipedia.org/wiki/Token_no_fungible
- https://ec.europa.eu/eip/agriculture/en/event/use-blockchain-technologies-agri-food-sector

5. Towards Federated Learning at Scale: System Design

a) Explica las principales ideas presentadas en este trabajo.

El trabajo presenta un informe sobre el funcionamiento del Federated Learning (en adelante FL) que es una técnica de aprendizaje para entrenar algoritmos de forma descentralizada utilizando como fuente de datos los almacenes locales de las aplicaciones que residen en los dispositivos. En este informe se centra el trabajo en el entorno Mobile, en concreto para Android, aunque se especifica que la arquitectura no depende de este entorno.

Se basa en el concepto de entrenar los modelos de forma distribuida y llevarlos hacia los dispositivos móviles (donde se encuentran esos datos) en contraposición a los sistemas actuales de Machine Learning (en adelante ML) que llevan los datos de entrenamiento a los servidores donde se encuentra el modelo a ejecutar.

El objetivo de este sistema es generalizar el uso del FL siguiendo los principios básicos que se describen en el informe sin restringir el uso del framework TensorFlow con ML.

El sistema se considera lo suficientemente maduro para poder llevarse a la práctica y resolver problemas de aprendizaje en decenas de millones de usuarios. Se anticipa un uso para miles de millones de dispositivos.

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

El aprendizaje colaborativo (FL) es una solución distribuida de machine Learning que permite entrenar un modelo a partir de datos completamente descentralizados, que se generan mediante aplicaciones en diferentes dispositivos (por ejemplo, los teléfonos móviles).

La arquitectura distribuida se basa en diferentes capas o niveles (servidor, dispositivos) y en cómo se comunican estas diferentes capas.

Servidor (FL Server)

El servidor se basa en un modelo de programación para comunicaciones concurrentes (Actor programming model) que utilizan mensajes.
Está formado por lo que se llaman "actores" que son procesos específicos que manejan un flujo de mensajes o eventos de forma secuencial.
Cada actor toma decisiones locales, envía mensajes a otros actores o crea más actores. Cada uno de ellos puede estar localizado de forma distribuida en diferentes máquinas en centros de datos diferentes.

Los actores que componen este modelo de programación son los siguientes:

- **Coordinador (actor o proceso principal):** se encarga de la sincronización y gestión de las rondas de procesamiento. Cada coordinador gestiona su propio conjunto de dispositivos (móviles) para el FL y es propietario de ese conjunto. Este coordinador le indica al selector cuántos dispositivos necesita y cuántos ha aceptado. Ese cálculo se realiza en función de las necesidades de las tareas FL que están programadas.

- **Agregadores:** son creados por el coordinador para administrar las tareas de cada ronda. A su vez, pueden generar más agregadores para delegar el trabajo en función de la cantidad de dispositivos que hay y del tamaño de la actualización. El agregador principal se denomina **Maestro Agregador.**

- **Selector:** su función principal es la de aceptar y reenviar las conexiones de los dispositivos a los agregadores. Reciben la información del coordinador periódicamente. Esa información les indica a los selectores que le envíen el conjunto de dispositivos conectados a los agregadores. Los selectores se distribuyen globalmente cerca de los dispositivos.

El servidor también requiere una capacidad de procesamiento que está determinada por el tamaño de la muestra de dispositivos que se ejecuta en cada ronda de entrenamiento (de cientos a miles de dispositivos), el procesamiento de los informes de actualizaciones que recibe de esos dispositivos (informes de resultados) y la gestión del tráfico donde debe gestionar los picos de conexión para diferentes franjas horarias.

Dispositivo

En este caso el dispositivo es un teléfono móvil en concreto para sistema operativo Android. La función principal del teléfono es mantener el almacén de datos locales para que se usen en el entrenamiento y evaluación del modelo.

Esta arquitectura se basa en el envío de modelos de entrenamiento a los dispositivos, que ejecutan las tareas de entrenamiento sobre los datos locales. Las aplicaciones instaladas en los dispositivos son las encargadas de generar los datos locales (almacenes de datos o Example store) y ponerlos a disposición del FL (usando una API que se les proporciona).

Los elementos que podemos identificar en la arquitectura del dispositivo son:

- **Aplicación:** Son las generan los datos que se almacenan en el almacén de datos y se registran en un FL Runtime. Las aplicaciones deben cumplir los estándares del mercado para resolver vulnerabilidades malware (entre otros problemas que puedan surgir de seguridad).

- **Almacén de datos (Example Store):** contiene todos los datos generados por la aplicación que se usan para el cálculo del modelo. Los datos no pueden ser ilimitados porque dependen de la capacidad del dispositivo y por eso se recomienda establecer un límite para la base de datos de la aplicación. Se recomienda eliminar datos obsoletos (se proporcionan utilidades para facilitar esta labor).

- **FL Runtime:** programa un trabajo usando el Jobscheduler de Android, que reserva una ventana de tiempo específica para informar al servidor de que está listo para recibir tareas. Estos trabajos se invocan siempre que el teléfono esté inactivo, cargándose y conectado a una red WIFI para no afectar a la experiencia
  Una vez que el dispositivo se ha seleccionado, recibe el plan, consulta el almacén de datos y realiza los cálculos. Finalmente, devuelve el informe con las actualizaciones al servidor y libera los recursos que estuviera utilizando.

El mecanismo de comunicación entre la aplicación el FL Runtime y el almacén de datos se realiza a través del AIDL IPC de Android. También soporta es multi-tenant, es decir, múltiples tareas y modelos en la misma aplicación o servicio. Por último, los dispositivos participan de forma anónima. No se verifica la identidad del usuario y se protege mediante Android Remote Attestation Mechanism para evitar ataques malware que puedan alterar los resultados (usando data-poisoning).

Modelo de comunicación

Las comunicaciones en el modelo colaborativo distribuido pueden basarse en comunicaciones síncronas o asíncronas.

Mientras que el Deep Learning (en adelante DL) se ha desarrollado con modelos de entrenamiento asíncronos, la tendencia actual es ir haciendo el entrenamiento masivo de forma síncrona. El sistema FL se centra en la comunicación síncrona con los dispositivos y en mitigar la sobrecarga que se pueda producir durante el proceso de sincronización. Se procesan grandes cantidades de algoritmos Averaging y SGD.

El protocolo de comunicación para FL es **síncrono**. Se ejecuta entre dispositivos móviles (Android) y un servicio distribuido en la nube llamado servidor FL.

Los dispositivos informan al servidor de que están listos para ejecutar una tarea FL. Cada tarea dispone de unos parámetros específicos para evaluar el modelo. Potencialmente hay decenas de miles de dispositivos que están informando sobre su disponibilidad de forma concurrente en la misma ventana de tiempo, el servidor selecciona un subconjunto de ellos y los invita a participar en una

tarea FL determinada. Este proceso se denomina "rendevous" (encuentro) entre el servidor, los dispositivos y la ronda de aprendizaje.

El servidor le dice al dispositivo qué cálculos debe realizar a través del plan FL (estructura de datos que incluye el grafo y las instrucciones para ejecutarlo). Cuando se establece la ronda, el servidor envía a cada dispositivo los parámetros globales y otros estados necesarios en forma de punto de chequeo (checkpoints)

Cada teléfono realiza sus cálculos en función de la etapa y los datos locales a procesar. Cuando acaba envía un nuevo checkpoint con los resultados al servidor. Cada checkpoint indica exactamente en qué momento de la ronda se encuentra. Finalmente, el servidor incorpora la actualización al estado global y se repite el proceso.

La ejecución del modelo se ejecuta en tres rondas. Cada ronda está compuesta de tres fases:

- **Selección:** durante esta fase, los dispositivos que cumplen con los criterios de selección establecidos (ya mencionados anteriormente) se registran en el servidor abriendo un canal bidireccional. El servidor selecciona un subconjunto de dispositivos, en función del número de dispositivos que necesita para el modelo y de criterios sobre los datos (por ejemplo, datos sobre las valoraciones de productos). Los teléfonos que no se seleccionan se descartan y se les informa de que pueden registrarse más tarde.
- **Configuración:** el servidor envía el plan FL, el checkpoint y el método de agregación.
- **Reporting:** El servidor espera a que los dispositivos envíen los informes con las actualizaciones (resultados). Cuando se reciben esas actualizaciones, si hay un número suficiente de ellas, la ronda se considera válida y entonces el servidor incorpora los resultados al modelo.

Tolerancia a fallos

El FL lleva incorporado un **mecanismo de tolerancia a los fallos**. Siempre se mantiene activo, aunque se produzcan errores y en el caso de que se produzca algún error, o bien la ronda continúa o se reinicia con los resultados de la ronda anterior.

Incluso aunque se pierda un proceso (actor) no impide que la ronda tenga éxito.

- **Fallo agregador o selector :** se pierden los dispositivos conectados a ese actor.
- **Fallo agregador maestro:** falla la ronda actual de la tarea FL, pero el coordinador la reinicia
- **Fallo coordinador:** el selector lo detecta y se recupera porque los coordinadores están registrados en un servicio de bloqueo compartido (sólo se da una única vez).

También se realizan tareas de análisis para vigilar el buen estado de los dispositivos y evitar así otros fallos que puedan darse en el sistema. Esos análisis registran el estado de los dispositivos (batería, ancho de banda, rendimiento etc.) y registran esta información en la nube (en ningún caso información contiene datos personales). De esta forma, se monitorizan los dispositivos y se generar alertas cuando se produce alguna desviación que comprometa a alguno de ellos, **realizando así de forma proactiva un control de errores**.

También se usan las métricas para poder definir de forma más aproximada el número de dispositivos a seleccionar. Se producen entre un 6-10% de desconexiones de los dispositivos producidos por diferentes tipos de fallos. Para compensar esta tasa, el servidor selecciona el 130% del número de

dispositivos. Estos datos pueden ir variando a medida que se va aumentando la población de dispositivos y según sean las métricas obtenidas, mejorando así los criterios de selección.

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

El modelo propuesto supone una gran mejora en los modelos de entrenamiento que existen en la actualidad. Aunque tiene menos rendimiento que los modelos centralizados, se compensa con los resultados obtenidos.

Uno de los ejemplos mencionados en el informe, es el uso de este modelo en el teclado móvil para Android (GBoard) que presenta al usuario predicciones sobre las búsquedas y uso de palabras clave, obtenidos mediante el entrenamiento de la red neuronal recurrente (RNN) con un modelo de 1,4 millones de parámetros.

El concepto de "llevar el modelo a los datos" supone un adelanto en la generación de modelos de entrenamiento porque utiliza datos de mayor valor añadido que los que residen en los servidores. Uno de los principales problemas de los modelos actuales es conseguir datos de calidad. De esta forma, al utilizar los datos generados por los dispositivos, éstos revelan comportamientos, predilecciones, uso de los usuarios de las aplicaciones etc., que contienen información más relevante que la que ofrecen los datos habituales que se obtienen a través de las conexiones vía Proxy.

Desde el punto de vista de la escalabilidad, la solución no se basa en el uso de lenguajes de programación específicos (Phyton) sino que se basa en la distribución de modelos de alto nivel basados en TensorFlow que independizan y facilitan el uso de este modelo para llevarlo a la práctica.

Sin embargo, el modelo tiene limitaciones en cuanto a la seguridad. Delega esta parte a las aplicaciones que generan los datos, limita el uso de Secure Aggregation a una ronda y sólo lo aplica en la fase de informes. Aunque la idea inicial es la de proporcionar una suite de herramientas que ayuden a preservar la privacidad y refuercen la protección de las actualizaciones, los costes crecen de forma cuadrática en función del número de usuarios.

Esta limitación lleva a que el modelo sea factible con Secure Aggregation únicamente para cientos de dispositivos, por ello esto sería un punto de mejora.

También tiene limitaciones en cuanto a la gestión de dispositivos, debe afrontar el reto de gestionar la disponibilidad por franja horaria de forma más eficiente y también la conectividad de dispositivos no confiables.

Por último, otro de los retos es el de gestionar las interrupciones en modo de ejecución y la limitación que existe en cuando al almacenamiento de cada dispositivo y de los recursos informáticos. Actualmente la selección de dispositivos se ha estandarizado para seleccionar móviles Android de 2 Gb y con versiones recientes. De momento, no se han detectado sesgos significativos causados por esta selección, pero es sistema debería afrontar el reto de incluir diferentes dispositivos con características diferentes.

En conclusión, considero que este sistema distribuido de aprendizaje puede tener aplicaciones prácticas muy importante en el ámbito de la salud (Federated Analysis). Hemos vivido una pandemia y se han tratado de recopilar datos de todo el mundo para aportar información a los sistemas de Big Data y ofrecer así datos relevantes que pudieran prevenir brotes nuevos.

El FL podría ser una solución para el problema inicial de acceso a todos esos datos (siempre que se garantice la seguridad de todos los usuarios).

Es una solución que democratiza el acceso a Big Data fuera de los ámbitos de las grandes corporaciones, universidades y organismos gubernamentales, ya que cualquier empresa podría desarrollar aplicaciones y usar estos servicios para acceder a modelos de ML sin necesidad de invertir en generar la infraestructura necesaria, así como la recopilación de datos, su limpieza y la compilación del modelo.

Fuentes:

- https://blog.ml.cmu.edu/2019/11/12/federated-learning-challenges-methods-and-future-directions/
- https://research.aimultiple.com/federated-learning/
- https://medium.com/bcggamma/federated-learning-the-next-big-step-ahead-for-data-sharing-2ae32d375309
- https://viso.ai/deep-learning/federated-learning/