Sistemas Distribuidos

Bloque II

Desarrollo de sistemas distribuidos y aplicaciones web.

Tema 3 .1

Fundamentos de las tecnologías de comunicación de aplicaciones

Nicolás Rodríguez
nicolas.rodriguez@urjc.es

Índice

- **Falacias de la computación distribuida**
  - Tolerancia a fallos
  - Sincronización
  - El Problema de los Dos Generales
  - El Problema de los Generales Bizantinos
- Replicación y particionamiento
- Consistencia

Falacias de la computación distribuida

- Las "Falacias de la Computación Distribuida" son un conjunto de suposiciones erróneas hechas por los programadores cuando se desarrollan sistemas distribuidos.
- Estas falacias pueden llevar a errores de diseño y problemas operativos.

Falacias de la computación distribuida

- La red es confiable: Suponer que la red siempre estará disponible y que no se perderán mensajes puede llevar a fallos graves.
- La latencia es cero: Ignorar el tiempo de transmisión puede causar problemas en el rendimiento y la sincronización.
- El ancho de banda es infinito: Suponer un ancho de banda ilimitado puede llevar a diseños que no escalan bien.
- La red es segura: Ignorar las preocupaciones de seguridad puede exponer el sistema a ataques.

Falacias de la computación distribuida

- La topología no cambia: Las redes cambian; los sistemas deben ser diseñados para adaptarse a estos cambios.
- Hay un administrador: Suponer que habrá una intervención manual en la operación de la red es poco realista en grandes sistemas.
- El coste de transporte es cero: Ignorar el coste de los datos puede llevar a soluciones ineficientes.
- La red es homogénea: Suponer que todo el sistema usa las mismas normas y protocolos es una simplificación excesiva.

Tolerancia a fallos – Introducción

- La tolerancia a fallos es un concepto clave en la computación distribuida y en el diseño de sistemas en general.
- Se refiere a la capacidad de un sistema para seguir operando de manera satisfactoria en la presencia de fallos parciales.
- Este concepto es crucial para garantizar la fiabilidad y la disponibilidad continua de un sistema, especialmente en entornos donde los fallos son inevitables debido a la complejidad del hardware, del software o de la red.

Tolerancia a fallos – Conceptos clave

- Detección de fallos: Identificar rápidamente las partes del sistema que han fallado.
- Aislamiento de fallos: Limitar los efectos del fallo a la menor parte posible del sistema para evitar una falla total.
- Recuperación de fallos: Restaurar la operación normal del sistema después de un fallo, ya sea mediante la reanudación del componente fallido o su sustitución.

Tolerancia a fallos – Conceptos clave

- Redundancia: Tener componentes o sistemas duplicados para que, en caso de fallo, otro pueda tomar el relevo. La redundancia puede ser:
  - De hardware (servidores duplicados).
  - De software (múltiples instancias de un servicio) o
  - De datos (como replicación de bases de datos).
- Reintento: Capacidad para intentar nuevamente operaciones que han fallado y manejar errores de manera ágil.
- Persistencia del estado: Diseñar sistemas sin estado facilita la recuperación, mientras que, en sistemas con estado, es crucial asegurar la persistencia y coherencia de este.

Tolerancia a fallos – Importancia

- Disponibilidad: Un sistema tolerante a fallos puede ofrecer una alta disponibilidad, lo cual es esencial para servicios críticos y operaciones empresariales continuas.
- Fiabilidad: Aumenta la confianza en el sistema, ya que puede manejar fallos sin interrumpir el servicio.
- Seguridad: En algunos casos, la tolerancia a fallos también implica mantener la seguridad del sistema en presencia de componentes defectuosos.

Tolerancia a fallos – Desafíos

- Complejidad: Diseñar e implementar sistemas tolerantes a fallos puede ser complejo y costoso.
- Rendimiento: La redundancia y los mecanismos de recuperación pueden afectar el rendimiento del sistema.
- Pruebas: Probar completamente la tolerancia a fallos puede ser difícil, ya que implica simular una variedad de fallos.

Tolerancia a fallos – Ejemplos

- Sistemas de Bases de Datos: Usan replicación y transacciones para asegurar la integridad de los datos.
- Sistemas de Archivos Distribuidos: Como HDFS, que almacena múltiples copias de datos para proteger contra la pérdida de datos.
- Infraestructura de Red: Diseñada para re-rutear el tráfico en caso de fallo de un enlace.

Sincronización – Introducción

- La sincronización en sistemas distribuidos es un desafío clave que implica:
  - Coordinar acciones.
  - Mantener la consistencia de los datos a través de múltiples nodos.
  - Estos no están necesariamente operando al unísono.
- En sistemas distribuidos, los nodos pueden estar separados físicamente y comunicarse a través de una red, lo que introduce incertidumbre y retardo en la comunicación.

Sincronización – Conceptos clave

- Sincronización de relojes:
  - Mantener los relojes de los diferentes nodos sincronizados es crucial para operaciones que dependen del tiempo.
  - Algoritmos como NTP (Network Time Protocol) se utilizan para sincronizar relojes a una fuente de tiempo común.
- Sincronización de datos:
  - Asegurar que todos los nodos tengan una vista consistente de los datos.
  - Complejo debido a la latencia de la red y las actualizaciones concurrentes.
  - Mecanismos como bloqueo distribuido, transacciones distribuidas y protocolos de consenso (como Paxos o Raft) se utilizan para mantener la coherencia de los datos.
- Sincronización de procesos:
  - Coordinar la ejecución de procesos en diferentes nodos.
  - Esto incluye asegurarse de que las operaciones se realicen en un orden específico cuando sea necesario.

Sincronización – Desafíos

- Latencia de red: La comunicación a través de la red introduce retrasos, lo que puede llevar a inconsistencias.
- Fallos y particiones de red: Los fallos en los nodos o en la red pueden interrumpir los mecanismos de sincronización.
- Concurrencia: La gestión de acceso concurrente a recursos compartidos es compleja y requiere mecanismos de sincronización.

Sincronización – Estrategias y mecanismos

- Algoritmos de Exclusión Mutua Distribuida: Garantizan que solo un nodo a la vez pueda realizar una operación crítica.
- Relojes Lógicos y Físicos: Los relojes lógicos (como los relojes de Lamport o relojes vectoriales) se utilizan para mantener un orden causal de eventos, mientras que los relojes físicos se sincronizan para reflejar el tiempo real.
- TrueTime de Google: permite un nivel de consistencia y coordinación que es difícil de lograr en sistemas distribuidos tradicionales

Sincronización – Ejemplos

- Bases de Datos Distribuidas: Mantienen la consistencia de los datos a través de transacciones distribuidas y protocolos de replicación.
- Sistemas de Archivos Distribuidos: Garantizan la coherencia de los datos entre diferentes nodos.
- Computación en la Nube y Microservicios: Coordinan múltiples servicios y procesos que se ejecutan en paralelo y a menudo de forma independiente.

El Problema de los Dos Generales

- Escenario:
  - El problema describe dos generales que planean atacar una ciudad fortificada.
  - Están acampados en colinas separadas, con la ciudad en el valle entre ellos.
- Desafío de comunicación:
  - Para tener éxito, deben atacar simultáneamente.
  - Sin embargo, la única forma de comunicarse es a través de mensajeros que deben atravesar el valle, donde hay riesgo de ser capturados por el enemigo.
- Dilema central:
  - Cada general no puede estar seguro de que sus mensajes de confirmación para coordinar el ataque hayan sido recibidos.
  - Un general envía un mensajero para proponer un plan de ataque, pero no puede estar seguro de que el mensaje haya llegado.
  - Si recibe una confirmación, no puede estar seguro de que su confirmación de la confirmación haya llegado, y así sucesivamente.

El Problema de los Dos Generales

- Confianza en la comunicación:
  - El problema resalta la imposibilidad de garantizar la fiabilidad absoluta en la comunicación sobre canales no fiables.
  - Es especialmente relevante en SSDD donde la comunicación entre nodos puede ser incierta.
- Acuerdo y consenso: Muestra las dificultades para alcanzar un consenso o acuerdo en la presencia de incertidumbre comunicacional, lo que es un aspecto fundamental en la computación distribuida.

El Problema de los Generales Bizantinos

- Escenario:
  - Un grupo de generales bizantinos, cada uno al mando de una parte del ejército bizantino, que deben acordar un plan común de acción (como atacar o retirarse) frente a una ciudad enemiga.
- Comunicación:
  - Los generales están separados y deben comunicar sus planes a través de mensajeros.
- Desafío:
  - Al menos uno de los generales (o más) podría ser un traidor que intentará confundir a los demás para que el plan falle.
  - El desafío es llegar a un consenso en presencia de estos elementos no confiables.

El Problema de los Generales Bizantinos

- Confianza y traición: El problema resalta la dificultad de alcanzar un consenso en un sistema distribuido cuando hay componentes no confiables que pueden proporcionar información falsa o contradictoria.
- Tolerancia a Fallos Bizantinos: Esto llevó al concepto de Tolerancia a Fallos Bizantinos (BFT), que se refiere a la capacidad de un sistema para continuar operando correctamente incluso en presencia de nodos defectuosos o malintencionados.

El Problema de los Generales Bizantinos

- Algoritmos de Consenso Bizantino:
  - Se han desarrollado varios algoritmos para tratar de resolver este problema en la práctica, como el algoritmo PBFT (Practical Byzantine Fault Tolerance).
  - Estos algoritmos son fundamentales en áreas como las redes de blockchain y criptomonedas, donde la confiabilidad y la seguridad son cruciales.
- Complejidad y recursos: Los algoritmos que abordan la Tolerancia a Fallos Bizantinos suelen:
  - Ser complejos.
  - Requerir mayor comunicación y coordinación entre los nodos.
  - Como consecuencia, aumentar el uso de recursos y la latencia.

Índice

- Falacias de la computación distribuida
- **Replicación y particionado**
  - Replicación
  - Particionado
  - Transacciones
- Consistencia

Replicación y particionado – Introducción

- La replicación y el particionamiento son técnicas fundamentales en el diseño de bases de datos distribuidas y sistemas de almacenamiento.
- Estas estrategias se utilizan para mejorar el rendimiento, la disponibilidad y la escalabilidad de los sistemas, así como para asegurar la resistencia a fallos.
- La replicación implica mantener copias de los mismos datos en múltiples nodos o servidores. Esto mejora la disponibilidad y la resistencia a fallos, pero introduce desafíos en mantener la consistencia de los datos.
- El particionamiento divide los datos en diferentes nodos para mejorar la escalabilidad y el rendimiento.

Replicación

- Replicación basada en el líder
  - Un nodo (líder) maneja todas las escrituras y actualizaciones. Las replicas (seguidores) se sincronizan con el líder.
  - Común en sistemas que requieren fuerte consistencia y en situaciones donde es aceptable tener un punto único de escritura.
- Replicación multilíder
  - Varios nodos actúan como líderes, permitiendo escrituras simultáneas en diferentes nodos.
  - Útil para mejorar la escritura y la disponibilidad de datos en múltiples regiones geográficas, aunque puede complicar la resolución de conflictos.
- Replicación sin líderes (quorum)
  - Las escrituras y lecturas se hacen en un conjunto de nodos, y se requiere un quórum (mayoría) para considerar una operación exitosa.
  - Equilibra la carga y aumenta la disponibilidad, pero puede ser más complejo manejar la consistencia.

Particionado

- Por Clave-Valor
  - Los datos se particionan basándose en las claves de los registros. Cada partición contiene un rango específico de claves.
  - Simplifica el acceso y la distribución de los datos, pero puede llevar a particiones desequilibradas si las claves no están bien distribuidas.
- Índices secundarios
  - En sistemas con índices secundarios, el particionamiento se vuelve más complejo, ya que los índices pueden apuntar a registros en diferentes particiones.
  - Necesario en consultas complejas, pero puede aumentar la complejidad del sistema y afectar el rendimiento.
- Rebalanceado
  - Implica mover datos entre nodos para mantener un balance de carga. Es esencial en sistemas dinámicos donde la carga o el tamaño de los datos cambian con el tiempo.
  - Mantiene el sistema equilibrado y eficiente, pero requiere mecanismos sofisticados para manejar el movimiento de datos sin afectar la disponibilidad.

Transacciones

- Las transacciones en sistemas informáticos, especialmente en bases de datos y sistemas distribuidos, son fundamentales para garantizar la integridad y coherencia de los datos.
- Una transacción es una secuencia de operaciones que se tratan como una única unidad lógica de trabajo.
- En bases de datos, las transacciones aseguran que las operaciones sobre los datos se realicen de manera segura y confiable, incluso en presencia de fallos y errores.

Transacciones – Propiedades

- Atomicidad:
  - Garantiza que todas las operaciones en la transacción se ejecutan o ninguna se ejecuta.
  - Si una parte de la transacción falla, el sistema revierte todas las operaciones de la transacción.
- Consistencia:
  - Asegura que la transacción lleva la base de datos de un estado consistente a otro.
  - No se deben violar las restricciones de integridad de la base de datos.
- Aislamiento:
  - Determina cómo y cuándo los cambios realizados por una transacción son visibles para otras transacciones.
  - El aislamiento ayuda a prevenir conflictos de concurrencia.
- Durabilidad:
  - Una vez que una transacción se ha comprometido, sus cambios son permanentes, incluso en caso de fallo del sistema.

Índice

- Falacias de la computación distribuida
- Replicación y particionamiento
- **Consistencia**
  - Linealizabilidad
  - Consistencia causal
  - Teorema CAP
  - 2FC
  - Teorema FLP
  - Algoritmos de Consenso

Linealizabilidad – Introducción

- Es el tipo más fuerte de consistencia que un sistema distribuido puede ofrecer y juega un papel crucial en garantizar la fiabilidad y la previsibilidad de las operaciones en dichos sistemas.
- Es una propiedad de los SSDD en la que las operaciones realizadas parecen haber ocurrido en un orden secuencial único, incluso si se ejecutan en paralelo en diferentes nodos.
- Una serie de operaciones parece ejecutarse instantáneamente en un punto entre su inicio y finalización. Esto significa que el sistema debería comportarse como si cada operación se ejecutara una tras otra, sin solapamiento.

Linealizabilidad – Conceptos clave

- Orden Global: Existe un orden global de todas las operaciones que es consistente con el orden en que se ejecutaron en cada nodo individual.
- Predictibilidad: Los usuarios pueden predecir el resultado de sus operaciones basándose en el conocimiento de operaciones anteriores.
- Consistencia a través de nodos: Los cambios realizados en un nodo son visibles para todos los demás nodos en un orden que refleja la secuencia real de operaciones.

Consistencia causal – Introducción

- Es un modelo de consistencia en SSDD que relaja algunas de las restricciones de los modelos de consistencia más estrictos, como la linealizabilidad
- El objetivo es mejorar el rendimiento y la escalabilidad.
- A diferencia de la consistencia fuerte, que puede ser costosa en términos de rendimiento y no siempre es necesaria, la consistencia causal se centra en mantener un orden lógico de las operaciones que refleja las causas y los efectos dentro del sistema.

Consistencia causal – Conceptos clave

- Orden Causal:
  - Asegura que si una operación A "causa" otra operación B en cualquier nodo del sistema, entonces todos los nodos verán A antes que B. Esto significa que la secuencia de eventos respeta la causalidad lógica.
- Independencia de operaciones no relacionadas:
  - Operaciones que no están causalmente relacionadas pueden ser vistas en un orden diferente por diferentes nodos.
  - Esto permite más flexibilidad y un mejor rendimiento en comparación con la consistencia fuerte.
- Aplicabilidad:
  - En sistemas donde el mantenimiento de un orden global exacto es menos crítico, como en redes sociales, sistemas de documentos colaborativos, y sistemas de mensajería.
- Diseño del Sistema:
  - Determinar las dependencias causales entre operaciones puede ser complicado.
- Compromiso:
  - Necesidad de equilibrar un orden coherente con los beneficios del rendimiento y la escalabilidad.

Teorema CAP – Introducción

- El Teorema CAP, también conocido como el Principio de Brewer, es un concepto fundamental en la teoría de sistemas distribuidos.
- Formulado por Eric Brewer en 2000, el teorema establece que en cualquier sistema de computación distribuido, solo se pueden garantizar dos de las siguientes tres propiedades al mismo tiempo:
  - Consistencia (Consistency)
  - Disponibilidad (Availability)
  - Tolerancia a Particiones (Partition Tolerance)

Teorema CAP – Conceptos clave

- Consistencia (Consistency)
  - Cada lectura recibe la escritura más reciente o un error.
  - Significa que todos los nodos ven los mismos datos al mismo tiempo.
  - No hay discrepancias en los datos entre los distintos nodos del sistema, incluso después de una actualización de datos.
  - Por ejemplo, en una base de datos distribuida, después de que un dato es actualizado, todas las futuras lecturas reflejarán ese cambio (o devolverán un error si el cambio no se puede reflejar)
- Disponibilidad (Availability)
  - Cada solicitud recibe una respuesta, sin garantizar que contenga la última versión del dato.
  - Significa que el sistema siempre responde a las solicitudes de los clientes, aunque esos datos puedan no ser los más recientes.
  - Por ejemplo, un sistema de base de datos distribuida responde a todas las consultas de lectura/escritura, pero los datos que devuelve pueden no ser los más actualizados si parte del sistema está desconectado.

Teorema CAP – Conceptos clave

- Tolerancia a Particiones (Partition Tolerance)
  - El sistema sigue funcionando a pesar de la pérdida arbitraria de mensajes o fallos en parte del sistema.
  - Significa que el sistema puede seguir operando incluso si hay fallas de red que impiden que algunos nodos se comuniquen entre sí.
  - Por ejemplo, en una red distribuida, si un segmento se desconecta, los nodos restantes pueden seguir operando y procesando solicitudes.
- Implicaciones del Teorema CAP
  - Elección de diseño: No es posible diseñar un sistema distribuido que garantice simultáneamente todas estas tres propiedades al 100%.
  - Realidad de las particiones: Las particiones son inevitables en cualquier red distribuida, por lo que la tolerancia a particiones suele ser una necesidad.
  - Escenarios de aplicación: El teorema CAP es especialmente relevante en el diseño de bases de datos distribuidas y sistemas de almacenamiento de datos, donde la gestión de la consistencia y la disponibilidad son críticas.

Transacción en Dos Fases – Introducción

- La Transacción en Dos Fases (2PC, por sus siglas en inglés) es un protocolo de control de transacciones distribuidas utilizado en BBDD y SSDD para garantizar la atomicidad en transacciones que involucran múltiples nodos o recursos.
- El 2PC es fundamental para mantener la integridad de los datos en sistemas distribuidos donde las operaciones deben ser atómicas (es decir, todas las operaciones en la transacción deben completarse con éxito o, en caso de fallo, ninguna debe aplicarse).

Transacción en Dos Fases – Funcionamiento

- Fase de votación (Preparación)
  - Coordinador: Un nodo actúa como coordinador de la transacción y envía un mensaje de "preparación" a todos los participantes (nodos implicados en la transacción).
  - Participantes: Cada participante ejecuta la transacción hasta el punto de compromiso, pero no realiza cambios permanentes. Luego, vota "sí" (preparado) si puede comprometer la transacción o "no" si no puede.
  - Respuesta: Los participantes envían su voto al coordinador.
- Fase de compromiso
  - Decisión basada en Votos: Si todos los participantes votan "sí", el coordinador envía un mensaje de "compromiso" a todos los participantes. Si alguno vota "no", el coordinador envía un mensaje de "abortar".
  - Acción de los participantes: Después de recibir el mensaje de compromiso, los participantes completan la operación y liberan los recursos y bloqueos que mantenían. Si reciben un mensaje de abortar, deshacen las operaciones realizadas en la fase de preparación.
  - Confirmación de Compromiso/Aborto: Los participantes informan al coordinador una vez que han completado el compromiso o el aborto.

Transacción en Dos Fases – Conceptos clave

- Atomicidad: Asegura que todas las partes de la transacción se comprometen o ninguna se compromete, manteniendo la integridad de los datos.
- Bloqueo de recursos: Durante la fase de preparación, los recursos que están siendo utilizados por la transacción se bloquean, lo que puede llevar a bloqueos y problemas de rendimiento en sistemas de alta concurrencia.
- Resistencia a fallos: Aunque el 2PC puede manejar fallos, si el coordinador falla después de enviar mensajes de compromiso, pero antes de recibir confirmaciones, puede resultar en incertidumbre sobre el estado de la transacción.

Teorema FLP – Introducción

- El Teorema FLP, nombrado por sus autores Michael Fischer, Nancy Lynch y Michael Paterson, es fundamental en la teoría de SSDD.
- Presentado en 1985, el teorema aborda la imposibilidad de alcanzar el consenso en sistemas distribuidos asincrónicos si se permite incluso un único fallo.
- Se centra en sistemas distribuidos asincrónicos, donde no hay suposiciones sobre la velocidad de procesamiento o los tiempos de transmisión de mensajes.
- El teorema demuestra que, en tales sistemas, no es posible garantizar que todos los nodos alcancen el mismo consenso (decisión) en presencia de al menos un fallo de tipo "crash" (un nodo deja de funcionar).

Teorema FLP – Implicaciones

- Consenso determinista: El resultado principal es que no existe un algoritmo determinista que pueda garantizar el consenso en un sistema distribuido asincrónico si se permite la posibilidad de un fallo.
- Seguridad vs. Vivacidad: El teorema muestra que no se pueden garantizar simultáneamente la seguridad (todos los nodos no defectuosos llegan al mismo acuerdo) y la vivacidad (eventualmente se toma una decisión) en tales sistemas.

Teorema FLP – Importancia

- Desafío de diseño: El teorema FLP presenta un gran desafío en el diseño de algoritmos de consenso para sistemas distribuidos, especialmente aquellos que necesitan operar de manera confiable en presencia de fallos.
- Compromisos prácticos: En la práctica, los sistemas deben hacer compromisos, a menudo favoreciendo la vivacidad sobre la seguridad total o viceversa, dependiendo de los requisitos del sistema.

Algoritmos de consenso: Paxos – Introducción

- Paxos es un protocolo de consenso desarrollado por Leslie Lamport en los años 90, ampliamente reconocido y utilizado en SSDD.
- Su objetivo principal es alcanzar un acuerdo entre los nodos de un sistema distribuido, especialmente en situaciones donde los nodos pueden experimentar fallos.
- Paxos se divide en varias fases y roles.
- Roles en Paxos:
  - Proposers: Proponen valores a ser acordados por el sistema.
  - Acceptors: Votan para aceptar o rechazar propuestas.
  - Learners: Aprenden el valor acordado por los acceptors.

Algoritmos de consenso: Paxos – Fases

- Fase 1 - Preparación:
  - Un proposer genera una propuesta con un número de secuencia único.
  - Envía una solicitud a los acceptors para que acepten la propuesta.
  - Los acceptors responden, indicando si aceptarán la propuesta basándose en el número de secuencia.
- Fase 2 - Aceptación:
  - Si la mayoría de los acceptors aprueban la propuesta, el proposer envía un valor a ser aceptado junto con el número de propuesta.
  - Los acceptors votan sobre la propuesta final.
  - Si la mayoría acepta, el valor se considera elegido.
- Elección del Valor:
  - El valor elegido se comunica a los learners.
  - El sistema asegura que, a pesar de los fallos, un único valor consensuado se elija.

Algoritmos de consenso: Paxos – Conceptos clave

- Tolerancia a fallos: Paxos puede tolerar fallos de nodos hasta cierto punto sin perder la capacidad de alcanzar un consenso.
- Consenso distribuido: Permite a un conjunto de nodos, que pueden no estar siempre disponibles o ser confiables, llegar a un acuerdo sobre un valor específico.
- Robustez: Es resistente a problemas de red y fallos de nodos individuales.

Algoritmos de consenso: Paxos – Limitaciones

- Complejidad: Paxos es conocido por ser difícil de entender y de implementar correctamente.
- Rendimiento: Puede ser lento debido a la cantidad de rondas de comunicación necesarias, especialmente en redes con alta latencia o en sistemas con alta tasa de conflictos de propuestas.
- Adaptabilidad: Ajustar Paxos a casos de uso específicos o integrarlo en sistemas existentes puede ser complejo.

Algoritmos de consenso: Paxos – Aplicaciones

- Sistemas de almacenamiento distribuido: Como en sistemas de bases de datos distribuidas y sistemas de archivos distribuidos.
- Coordinación de servicios: Utilizado en sistemas de orquestación y coordinación como Apache ZooKeeper.
- Infraestructura de cloud computing: Para manejar la consistencia de datos y el estado en entornos de computación en la nube.

Algoritmos de consenso: Zab – Introducción

- Zab, que significa "Zookeeper Atomic Broadcast", es un protocolo de consenso para el sistema de coordinación de Apache ZooKeeper.
- ZooKeeper es un servicio centralizado para mantener la configuración de información, nombrar, proporcionar sincronización distribuida, y proporcionar servicios de agrupación en SSDD.
- Zab es fundamental para garantizar que ZooKeeper pueda manejar roles de manera confiable, especialmente en entornos donde se pueden experimentar fallos de nodos.
- Roles en Zab:
  - Líder: Maneja todas las solicitudes de escritura y coordina la actualización de los seguidores.
  - Seguidores: Mantienen una copia del estado y responden a las solicitudes de lectura.

Algoritmos de consenso: Zab – Fases

- Fase de descubrimiento: Cuando se elige un nuevo líder, establece el estado más actualizado del sistema.
- Fase de sincronización: El líder sincroniza su estado con los seguidores para asegurarse de que todos tengan los mismos datos.
- Fase de difusión: Una vez sincronizados, el líder puede difundir nuevas transacciones a los seguidores.
- Transacciones:
  - Cada cambio en el estado se maneja como una transacción.
  - Las transacciones se difunden de manera atómica y en orden.

Algoritmos de consenso: Zab – Conceptos clave

- Consistencia: Asegura que todos los nodos (seguidores) mantengan una copia consistente del estado.
- Recuperación de fallos: En caso de falla del líder, el protocolo puede elegir y cambiar a un nuevo líder, manteniendo la continuidad del servicio.
- Orden de entrega garantizado: Las transacciones se entregan en el mismo orden en todos los nodos.

Algoritmos de consenso: Zab – Limitaciones

- Dependencia del líder: El rendimiento y la disponibilidad del sistema dependen en gran medida del líder, lo que puede ser un punto único de fallo.
- Escalabilidad: Aunque Zab maneja bien los entornos distribuidos, la escalabilidad puede verse afectada en sistemas con un gran número de nodos debido a la sobrecarga de coordinación.

Algoritmos de consenso: Raft – Introducción

- Raft es un algoritmo de consenso para sistemas de computación distribuida que se centra en la comprensibilidad y la facilidad de implementación.
- Diseñado por Diego Ongaro y John Ousterhout en 2014, Raft ofrece una alternativa más accesible al complejo algoritmo Paxos, manteniendo un enfoque en la seguridad y eficiencia.
- https://thesecretlivesofdata.com/raft/

Algoritmos de consenso: Raft – Fases

Elección del Líder:

- Inicio de la Elección:
  - Si un nodo no recibe comunicación del líder en un tiempo determinado (denominado "tiempo de espera"), se convierte en candidato y comienza una elección.
- Votación:
  - El candidato solicita votos de otros nodos. Si recibe votos de la mayoría de los nodos, se convierte en el líder.
- Heartbeat:
  - Una vez elegido, el líder envía mensajes periódicos de "heartbeat" a todos los nodos para mantener su autoridad y prevenir nuevas elecciones.

Algoritmos de consenso: Raft – Fases

Replicación de log:

- Propuestas de Clientes:
  - El líder recibe propuestas de cambio de estado (como comandos de clientes) y las agrega a su log.
- Replicación:
  - El líder replica estas entradas de log a los demás nodos (seguidores).
- Compromiso:
  - Una vez que una entrada de log ha sido replicada en la mayoría de los nodos, se considera "comprometida" y se aplica al estado del sistema.

Algoritmos de consenso: Raft – Fases

Manejo de fallos y cambios de Líder:

- Fallos de Líder:
  - Si el líder falla, los nodos esperarán un tiempo de espera sin recibir el "heartbeat" y luego comenzarán una nueva elección.
- Consistencia de log:
  - Si los logs entre el líder y los seguidores difieren, el líder sobrescribe las entradas inconsistentes en los seguidores con las suyas para garantizar la consistencia.

Algoritmos de consenso: Raft – Conceptos clave

- División del problema: Raft separa claramente el problema del consenso en dos partes: la elección del líder y la replicación de log.
- Seguridad y robustez: Raft asegura que el log replicado sea consistente y que, una vez comprometida una entrada de log, permanezca como parte permanente del log.
- Comprensibilidad: Una de las metas principales de Raft es ser más fácil de entender y razonar que otros algoritmos de consenso.

Algoritmos de consenso: PoW – Introducción

- El "Proof of Work" (PoW), es un mecanismo fundamental utilizado en varios sistemas criptográficos, siendo más notablemente asociado con Bitcoin y otras criptomonedas.
- Su propósito principal es prevenir el spam y los ataques de denegación de servicio.

Algoritmos de consenso: PoW – Fases

- Desafío Criptográfico:
  - En PoW, se plantea un desafío criptográfico que requiere un esfuerzo significativo para resolverlo.
  - Este desafío generalmente involucra encontrar un valor (nonce) que, cuando se combina con los datos de la transacción y se pasa a través de una función hash criptográfica, produce un hash que cumple con ciertos criterios (por ejemplo, un número específico de ceros al principio).
- Minería:
  - Este proceso de resolver el desafío criptográfico se conoce como "minería".
  - Los mineros utilizan computadoras de alto rendimiento para realizar cálculos hash de manera repetitiva hasta que uno encuentra la solución correcta.
- Recompensa:
  - Una vez que un minero resuelve el desafío, presenta la solución a la red para verificación.
  - Si la solución es correcta y cumple con las reglas de la red, el minero es recompensado, por ejemplo, con bitcoins en el caso de la red Bitcoin.
- Creación de un Nuevo Bloque:
  - La solución se utiliza para validar un bloque de transacciones, que luego se agrega a la blockchain.

Algoritmos de consenso: PoW – Conceptos clave

- Seguridad: Dificulta la manipulación de la blockchain, ya que requiere una cantidad significativa de esfuerzo computacional para resolver los desafíos y validar transacciones.
- Descentralización: Permite a cualquier persona con el hardware necesario participar en el proceso de minería, manteniendo la red descentralizada.
- Prevención de doble gasto: Asegura que las mismas monedas digitales no se gasten más de una vez.

Algoritmos de consenso: PoW – Limitaciones

- Consumo de energía: PoW es intensivo en energía, lo que ha generado preocupaciones sobre su impacto ambiental, especialmente en redes como Bitcoin que requieren una gran cantidad de esfuerzo computacional.
- Centralización de la minería: La minería se ha vuelto cada vez más centralizada debido a la necesidad de hardware especializado y acceso a electricidad barata.
- Escalabilidad: La velocidad a la que se pueden agregar nuevos bloques y procesar transacciones es limitada, lo que plantea desafíos de escalabilidad.

Algoritmos de consenso: PoW – Introducción

- Proof of Stake (PoS), es un mecanismo alternativo al "Proof of Work" (PoW) para alcanzar el consenso en las redes de blockchain.
- Fue introducido como una solución a algunos de los problemas asociados con el PoW, particularmente el alto consumo de energía.
- En PoS, la capacidad de un nodo para validar bloques de transacciones se basa en la cantidad de moneda (o 'stake') que tiene en la red, en lugar de su capacidad para resolver desafíos criptográficos como en PoW.

Algoritmos de consenso: PoW – Fases

- Selección de validadores:
  - Los validadores son seleccionados para crear un nuevo bloque, basándose en su participación en la red. Esta participación se mide generalmente por la cantidad de moneda que poseen y, en algunos casos, por el tiempo durante el cual la han mantenido.
- Validación de bloques:
  - Los validadores son responsables de verificar las transacciones y añadir nuevas a la blockchain.
  - A diferencia del PoW, no hay un proceso de minería intensivo en recursos.
- Recompensas:
  - Los validadores son recompensados con tarifas de transacción o, en algunos casos, con nuevas monedas.
  - No se requiere un hardware especializado, lo que reduce significativamente el consumo de energía en comparación con PoW.

Algoritmos de consenso: PoW – Ventajas

- Eficiencia energética: Elimina la necesidad de poder computacional intensivo y, por lo tanto, consume mucha menos energía que PoW.
- Menor riesgo de centralización: Dado que no se requiere equipo especializado, existe un menor riesgo de que la minería se centralice en aquellos con recursos para comprar hardware especializado.
- Seguridad: Algunas variantes de PoS incorporan medidas adicionales para garantizar la seguridad y reducir la posibilidad de ataques maliciosos, como el "Nothing at Stake".

Algoritmos de consenso: PoW – Desventajas

- Riqueza = Poder: Una crítica es que PoS favorece a aquellos que ya tienen grandes cantidades de moneda, potencialmente llevando a una centralización del poder financiero.
- Nada en Juego: Existe el desafío del problema de "Nothing at Stake", donde los validadores pueden querer validar en múltiples cadenas de bloques para maximizar sus recompensas, aunque esto se aborda en diseños más recientes de PoS.
- Ataques del 51%: Mientras que en PoW un ataque del 51% requiere controlar el 51% del esfuerzo de cómputo, en PoS, esto implicaría poseer el 51% de la moneda, lo cual es teóricamente más difícil de lograr.

Sistemas Distribuidos

Bloque II

Desarrollo de sistemas distribuidos y aplicaciones web.

Tema 3 .1

Fundamentos de las tecnologías de comunicación de aplicaciones

URJC Escuela Técnica Superior de Ingeniería Informática

Nicolás Rodríguez
nicolas.rodriguez@urjc.es