Informe del proyecto distribuido

En nuestro sistema, se distinguen dos roles fundamentales: servidores y clientes. Los clientes desempeñan un papel específico orientado únicamente a la realización de

consultas hacia el sistema, sin participar en la ejecución directa de operaciones complejas. Todas las funcionalidades operativas y de gestión de datos son llevadas a cabo exclusivamente por los servidores. En este informe técnico, el foco de análisis y descripción se concentra en detallar el diseño, la estructura organizativa y las responsabilidades inherentes de los servidores dentro del entorno del sistema distribuido.

Arquitectura

El sistema utiliza Chord, un protocolo de enrutamiento y búsqueda basado en Distributed Hash Table (DHT). Este organiza los nodos en un anillo lógico y distribuye claves entre ellos para lograr una red escalable, eficiente y descentralizada.

Organización de su sistema distribuido

El sistema distribuido está organizado siguiendo el protocolo Chord, en el cual los nodos están estructurados en un anillo lógico. Cada nodo tiene un identificador único generado a partir de una función hash y es responsable de gestionar un rango específico de claves. La asignación de claves a nodos se realiza utilizando una función hash uniforme, y las búsquedas de datos son redirigidas eficientemente mediante las tablas de finger que mantienen referencias a otros nodos en el anillo. Este enfoque garantiza una distribución equilibrada de las claves, escalabilidad y tolerancia a fallos.

Roles de su sistema

Todos los nodos (servidores) tienen roles idénticos, actuando como administradores de datos, enrutadores de consultas y participantes en el mantenimiento del anillo lógico. Cada nodo almacena los datos correspondientes a su rango de claves y responde a las solicitudes relacionadas. Además, utiliza su tabla de *finger* para reenviar consultas hacia el nodo sucesor adecuado, optimizando el tiempo de búsqueda. Los nodos también participan en procesos de estabilización y detección de fallos, ajustando dinámicamente la estructura del anillo según sea necesario.

Distribución de servicios en ambas redes de docker

El sistema utiliza dos redes de Docker para distribuir los servicios: una en la que estarán alojados los clientes, los cuales harán las consultas teniendo en cuenta la API proporcionada por los servidores, y otra donde se encontrarán dichos servidores para comunicación, enrutamiento y almacenamiento de datos entre ellos. En la red de los servidores, los servicios ejecutan el protocolo Chord, manejando la enrutación de consultas y el intercambio de mensajes entre nodos. Cada uno mantiene una base de datos local que almacena los datos correspondientes a su rango de claves.

Procesos o el problema de cuántos programas o servicios posee el sistema

Tipos de Procesos Dentro del Sistema y su organización

En el sistema se identifican tres tipos principales de procesos:

- **Creación de Agentes y Funcionalidades:**

  Estos procesos se encargan de almacenar nuevos agentes y funcionalidades en el sistema. Inicialmente no hay un servidor relacionado a esa información, por lo que es necesario generar el hash para saber en que servidor hacer la operación.

- **Ejecución de Soluciones con código nativo:**

  Este proceso no es necesario que sea ejecutado por un servidor o un conjunto de servidores específicos, ya que todos están capacitados para ejecutarlo. El servidor que recibe la solicitud es el encargado de procesarla.

- **Ejecución de Soluciones Basadas en Agentes:**

  Estos procesos ejecutan las soluciones que involucran a los agentes creados. Es ejecutado por el servidor que recibe la solicitud y este enciará solicitudes por el anillo de chord para obtener los agentes que no tenga en su base de datos para ejecutarlos

- **Consultas y Modificaciones por parte de Clientes:**

Los clientes pueden realizar consultas sobre los agentes y las funcionalidades existentes, así como modificar o eliminar ambos. En este caso la solicitud llega y si el servidor que la recibe no la puede contestar, esta recorre el anillo hasta dar con el servidor que está capacitado para responderla.

Tipo de Patrón de Diseño con Respecto al Desempeño

El sistema distribuido adoptará el **Patrón de Computación Distribuida** apoyándose en Chord para la gestión eficiente de recursos, asegurando un desempeño óptimo mediante el uso de programación asincrónica (async) para operaciones I/O-bound, hilos para tareas CPU-bound y procesos para tareas aisladas. Esta combinación permite la ejecución concurrente de múltiples tareas, balanceo de carga eficiente y alta tolerancia a fallos, evitando los cuellos de botella y mejorando la escalabilidad y resiliencia del sistema.

Comunicación o el problema de como enviar información mediante la red

En el diseño de este sistema distribuido, la comunicación entre los componentes (clientes y servidores) debe ser eficiente, confiable y escalable, considerando que los clientes y servidores residen en redes separadas conectadas mediante un router. El principal desafío es garantizar que los mensajes enviados entre los nodos del sistema lleguen de manera consistente, incluso en un entorno donde los servidores son iguales en jerarquía y utilizan el protocolo Chord para organizarse. Además, se requiere manejar el enrutamiento a través del router de manera efectiva, asegurando que las solicitudes de los clientes se distribuyan correctamente entre los servidores, y que los servidores puedan comunicarse entre sí para mantener la estructura lógica del anillo de Chord.

Tipo de comunicación: RPC y REST

El sistema emplea REST (Representational State Transfer) para la comunicación entre los clientes y los servidores, ya que este enfoque simplifica la interacción al exponer una API estandarizada que los clientes pueden consumir mediante solicitudes HTTP.

Esto facilita el desarrollo y la integración, ya que los clientes solo necesitan realizar solicitudes GET, POST u otras operaciones HTTP para interactuar con los servidores. Por otro lado, la comunicación entre servidores utiliza RPC (Remote Procedure Call), dado que es adecuado para las interacciones frecuentes y de baja latencia requeridas por el protocolo Chord, como la búsqueda de claves, la actualización de tablas de finger y la estabilización del anillo lógico.

Comunicación cliente-servidor y servidor-servidor

La comunicación cliente-servidor es unidireccional y basada en solicitudes. Los clientes envían consultas específicas a través de la API REST expuesta por los servidores y reciben respuestas con los datos procesados. Esta separación asegura que los clientes permanezcan independientes de la lógica interna de los servidores y evita complejidad adicional. En contraste, la comunicación entre servidores es bidireccional y sigue el esquema del protocolo Chord. Los servidores utilizan RPC para localizar claves en el anillo, actualizar referencias en sus tablas de finger y realizar operaciones de estabilización para garantizar la consistencia del sistema. Esto permite que el sistema distribuido sea autónomo y que los servidores colaboren entre sí sin depender de un nodo central.

Comunicación entre procesos

Dentro de cada servidor, la comunicación entre procesos asegura que las operaciones internas, como la gestión de datos y el manejo de solicitudes entrantes, se realicen de manera eficiente. Esto se logra mediante la utilización de colas de mensajes internas o comunicación basada en subprocesos (threads), dependiendo de las necesidades específicas del servidor. Este enfoque garantiza que los servidores puedan procesar solicitudes de clientes y coordinarse con otros servidores en el anillo de manera simultánea, sin interrupciones ni bloqueos.

Coordinación o el problema de poner todos los servicios de acuerdo

La estructura descentralizada de Chord permite que los servidores coordinen sus operaciones de manera eficiente sin necesidad de un nodo central. Las operaciones como la búsqueda de claves, la incorporación o eliminación de nodos, y la actualización de referencias en las tablas de finger requieren un protocolo bien definido para mantener la consistencia del sistema. Cada nodo actúa de manera independiente, pero las reglas del protocolo aseguran que todos los servidores lleguen a un consenso lógico sobre la ubicación y el estado de las claves.

Sincronización de acciones

Durante la estabilización del anillo lógico, cada servidor involucrado en una solicitud del cliente que modifique su base de datos sincroniza su información y la de los sucesores para garantizar la conectividad del anillo. Este proceso se realiza periódicamente y no requiere sincronización estricta en tiempo real, ya que Chord está diseñado para tolerar fallos temporales y estados inconsistentes a corto plazo. La sincronización se realiza mediante mensajes de consulta y respuesta entre nodos, asegurando que las actualizaciones se propaguen gradualmente a todo el sistema.

Acceso exclusivo a recursos y condiciones de carrera

Dado que los servidores pueden recibir solicitudes concurrentes tanto de clientes como de otros servidores, es fundamental evitar condiciones de carrera en el acceso a recursos compartidos, como datos almacenados localmente o la actualización de la tabla de finger. Esto se logra mediante mecanismos de exclusión mutua, como el uso de bloqueos (locks) o semáforos. De esta manera, se asegura que las modificaciones sean consistentes y que no ocurran errores debidos a accesos simultáneos no controlados.

Toma de decisiones distribuidas

La toma de decisiones en el sistema es completamente distribuida y sigue las reglas del protocolo Chord. Por ejemplo, al incorporar un nuevo nodo, este debe decidir su posición en el anillo basándose en el hash de su identificador y actualizar las referencias necesarias con sus nodos sucesor y predecesor. Este proceso no requiere la intervención de un coordinador central, ya que los nodos implicados negocian y

ajustan la estructura del anillo de manera local. De manera similar, en caso de fallos o desconexión de un nodo, los servidores restantes toman decisiones para reconfigurar el anillo y garantizar que los datos en el servidor ausente sean distribuidos por el anillo, manteniendo que cada dato esté repetido una cierta cantidad de veces. Este enfoque descentralizado asegura que el sistema sea escalable, tolerante a fallos y eficiente en la toma de decisiones.

Nombrado y Localización o el problema de dónde se encuentra un recurso y cómo llegar al mismo

El sistema implementa un esquema de nombrado basado en hashes para identificar y localizar recursos dentro de la red distribuida. Cada recurso se asocia con una clave generada mediante una función hash uniforme (como *SHA-1*). Esta clave es utilizada para determinar el nodo responsable del recurso en el anillo lógico. Los nodos también tienen identificadores únicos generados por la misma función hash, lo que permite una correspondencia directa entre claves y nodos. Para localizar un recurso, un nodo inicia una consulta que se propaga a través del anillo utilizando su tabla de finger, optimizando el tiempo de búsqueda a $O(log\ N)$, donde N es el número de nodos en el sistema.

Identificación de los datos y servicios

Los datos y servicios en el sistema están identificados mediante claves *hash* únicas. Estas claves se generan a partir de las propiedades de los datos o del nombre del servicio, asegurando una distribución uniforme en el anillo lógico. Este enfoque garantiza que cada nodo conozca qué claves está gestionando, evitando colisiones y redundancias innecesarias.

Ubicación de los datos y servicios

La ubicación de los datos y servicios en el sistema depende de las reglas del protocolo Chord. Cada nodo en el anillo es responsable de las claves que caen en el rango entre su identificador y el identificador de su predecesor. Esta distribución permite que los

datos y servicios se almacenen de manera distribuida y equilibrada entre los nodos, evitando sobrecarga en un único servidor.

Localización de los datos y servicios

La localización de datos y servicios se realiza mediante consultas distribuidas que utilizan las tablas de finger de cada nodo. Cuando un nodo recibe una solicitud para una clave específica, consulta su tabla de finger para reenviar la solicitud al nodo más cercano según el identificador de la clave. Este proceso se repite hasta que se alcanza el nodo responsable. Este mecanismo no solo garantiza la correcta localización de datos y servicios, sino que también optimiza el número de saltos necesarios, minimizando la latencia y maximizando la eficiencia del sistema.

Consistencia y Replicación

En sistemas distribuidos, la consistencia y replicación son fundamentales para garantizar que los datos se mantengan correctos y accesibles, incluso en presencia de fallos. Al tener múltiples nodos que pueden contener réplicas del mismo dato, es necesario resolver problemas relacionados con la sincronización, actualización y distribución de estas copias para evitar inconsistencias.

Distribución de los Datos

En Chord, los datos se distribuyen utilizando una Distributed Hash Table (DHT). Cada dato se asocia a una clave generada mediante una función hash uniforme. El nodo responsable de almacenar un dato es el sucesor más cercano de esa clave en el anillo lógico. Esta estrategia asegura una distribución equitativa de los datos entre los nodos, evitando cuellos de botella y sobrecarga en nodos específicos. Además, la redistribución automática de datos ocurre cuando los nodos se unen o salen del anillo, garantizando un equilibrio dinámico en el sistema.

Replicación y Cantidad de Réplicas

Para garantizar la disponibilidad y confiabilidad de los datos, Chord implementa un esquema de replicación en el que los datos se almacenan no solo en el nodo

responsable, sino también en sus k sucesores más cercanos en el anillo. Este número (k) determina el nivel de redundancia en el sistema, y en el caso de nuestro sistema tenemos $k = 3$.

Confiabilidad de las Réplicas tras una Actualización

En el sistema, la confiabilidad de las réplicas tras una actualización se asegura mediante un mecanismo de propagación coordinada.

Cuando un cliente solicita la modificación de un dato, el nodo responsable, determinado por la función hash, realiza la actualización localmente y propaga los cambios a sus dos sucesores inmediatos en el anillo lógico, que almacenan las réplicas. Este proceso utiliza RPC para enviar la nueva versión del dato a los nodos sucesores, verificando que cada uno haya recibido y aplicado correctamente la actualización antes de completar la operación.

En el caso de que uno de los nodos sucesores esté inactivo o no confirme la actualización, el nodo responsable intenta reenviar los datos al siguiente nodo activo en la cadena de sucesores, asegurando que las réplicas requeridas estén siempre disponibles. Además, durante el proceso de estabilización del anillo, cualquier nodo que detecte la salida de un sucesor reasigna automáticamente las claves afectadas al nuevo nodo correspondiente en el anillo.

Tolerancia a fallas

La arquitectura del sistema se basa en un enfoque descentralizado, donde cada nodo es autónomo, pero trabaja de manera colaborativa para manejar los datos distribuidos. Esto permite que el sistema continúe funcionando incluso si una o varias componentes fallan (hasta 2 en este caso). Mediante replicación de datos, redistribución dinámica de claves y estabilización del anillo lógico, el sistema puede operar de manera confiable bajo condiciones adversas y recuperarse rápidamente de fallos.

Respuesta a Errores

Cuando uno o dos nodos en el anillo fallan o se desconectan, los nodos adyacentes detectan su ausencia durante las operaciones periódicas de estabilización. Todos los nodos ajustan sus referencias al sucesor y predecesor, manteniendo la estructura del anillo consistente. Las claves que estaban bajo la responsabilidad del o los nodos fallidos son reasignadas (por su sucesor que es quien las conoce) automáticamente a los sucesores inmediatos, asegurando que los datos permanezcan disponibles (con 2 réplicas). Si un cliente intenta acceder a un dato cuyo nodo responsable está caído, la consulta es redirigida al siguiente nodo que contiene una réplica válida. Este enfoque permite al sistema gestionar fallos de manera transparente para los usuarios y mantener la integridad de los servicios ofrecidos.

Nivel de Tolerancia a Fallos Esperado

El sistema está diseñado para tolerar fallos de hasta dos nodos consecutivos sin pérdida de datos ni interrupciones en los servicios, gracias a la replicación en los dos sucesores inmediatos de cada nodo. Este nivel de tolerancia asegura que, incluso en escenarios donde múltiples nodos se desconecten simultáneamente o de forma secuencial, las operaciones de lectura y escritura puedan completarse sin errores. En situaciones más extremas, el sistema prioriza la recuperación de datos y la reconfiguración del anillo para minimizar el impacto en los usuarios.

Seguridad o el problema de que tan vulnerable es su diseño

Seguridad con respecto a la comunicación

Para esta parte tenemos pensado hacer un cifrado de datos en tránsito utilizando protocolos como TLS/SSL para cifrar los datos mientras se transmiten entre los nodos. Esto asegura que cualquier dato interceptado sea ilegible para terceros.También quisieramos mantener la integridad de los mensajes utilizando firmas digitales y hash para garantizar que los mensajes no sean alterados durante la transmisión.

Seguridad con respecto al diseño

Como se mencionó anteriormente vamos a distribuir las claves entre los nodos para evitar que un solo nodo se convierta en un punto de fallo o un objetivo atractivo para los ataques.