Distributed Machine Learning (DML):

Introducción

DML: es un sistema que permite a los usuarios entrenar modelos de machine learning de forma distribuida y realizar predicciones utilizando modelos previamente entrenados.

El sistema expone una **API REST** que ofrece los servicios de entrenamiento y predicción, y además incluye una aplicación de consola que encapsula toda la lógica necesaria para que el funcionamiento del sistema sea completamente transparente para el usuario.

Alcance del Sistema

1. El sistema ofrece diversos modelos de machine learning para tareas de ***regresión*** y ***clasificación.***
2. Permite a los usuarios subir archivos ****.csv**** con los datasets de entrenamiento o predicción.
3. Los usuarios pueden crear procesos de entrenamiento indicando:
    - el tipo de entrenamiento (regresión o clasificación).
    - el dataset que se usará.
    - los modelos a entrenar.
4. El sistema permite consultar el estado de los entrenamientos en curso.
5. Los usuarios pueden ejecutar predicciones utilizando modelos previamente entrenados.
6. Se pueden consultar los resultados generados por las predicciones.
7. Los usuarios pueden descargar los modelos ya entrenados.

Especificaciones del Sistema:

El sistema utiliza una arquitectura modular y desacoplada basada en dos tipos de nodos (Workers y DataBases)

Workers:

Los nodos Worker son responsables de procesar las peticiones de los usuarios y ejecutar las tareas de entrenamiento y predicción de los modelos.

Cada Worker funciona de forma totalmente independiente, sin necesidad de comunicarse ni coordinarse con otros Workers. Su única interacción es con los nodos DataBase, que actúan como fuente de datos y como mecanismo de control del trabajo distribuido.

Procesos de los Workers:

Cada worker realiza 4 tipos de procesos distintos:

1. REST:

Son los procesos detrás de cada endpoint que expone en su API. Se encargan de procesar las peticiones que reciben de los clientes, inicializando datos y realizando las peticiones necesarias sobre los nodos DataBases.

2. GET_MODEL:

Proceso que constantemente verifica si tiene disponibilidad para entrenar o predecir parte de un modelo y en caso de tener le pide a un nodo **DataBase** un modelo a entrenar o predecir, además de los metadatos necesarios para cumplir la tarea (**batch, tipo de modelo, parámetros actuales, tipo de proceso**, etc).

Una vez tiene los datos necesarios inicia un proceso de **Training/Prediction** según sea necesario. Dicho proceso se ejecuta de manera independiente.

3. Training/Prediction:

Realiza una iteración del entrenamiento del modelo o la predicción sobre el batch de datos con que se inicializó, manda a actualizar los resultados o parámetros del modelo a uno de los DataBases activos.

4. Descubrimiento:

Hilo que se encargan de saber que nodos DataBase están activos a través del DNS.

Este hilo mantiene una tabla con todos los nodos del sistemas (incluso los que no están activos), y a cada uno le asocia una variable booleana que muestra si está o no activo en la misma red.

Por otra parte cada vez que le realiza una petición a un nodo DataBase actualiza el estado del mismo según si le respondió o no.

Tolerancia a fallas de los Workers:

Los workers no conocen ni necesitan conocer la existencia de otros workers. No comparten estado ni intercambian mensajes.

Esta independencia permite escalar horizontalmente simplemente agregando nuevos nodos, y a demás garantiza una tolerancia a fallas de K-1, donde K es a la cantidad de Workers que hay activos en la red. Además garantiza que no ocurran fallas ante particiones y reconexiones de red siempre que exista un worker activo.

Coordinación entre Workers:

Para evitar que dos workers en la misma red entrenen o predigan sobre el mismo modelo al mismo tiempo, el sistema utiliza un mecanismo de heartbeat basado en un campo llamado ***health***, almacenado en los nodos DataBase.

Su funcionamiento es el siguiente:

1. **Selección del modelo disponible**:
    - Los nodos DataBase llevan un registro del health de cada modelo.
    - Un worker puede solicitar al DataBase un modelo cuyo health sea de hace más de 20 segundos (Ese modelo se considera libre o abandonado).
2. **Heartbeat del worker**:
    - Una vez que un worker toma un modelo, envía cada 10 segundos una señal al DataBase informando que sigue activo.
    - El DataBase actualiza (health = instante_actual) en los dueños del dato.
3. **Detección de fallo o abandono**:
    - Si no se recibe un heartbeat en más de 20 segundos, el DataBase considera que el worker presentó algún tipo de error.
    - Ese modelo vuelve a estar disponible para que otro worker lo tome.
4. **Reasignación automática**:
    - Cualquier worker que pida un modelo libre podrá tomarlo inmediatamente.
    - No se necesitan bloqueos distribuidos ni coordinación entre workers.

Ventajas del Mecanismo:

- Evita conflictos: ningún modelo será entrenado por dos workers simultáneamente.
- Alta disponibilidad: si un worker se cae, el trabajo se reasigna automáticamente.
- Escalabilidad simple: más workers = más capacidad de entrenamiento sin cambios estructurales.
- Desacoplamiento total: la lógica distribuida se delega a los nodos DataBase mediante el campo health.

DataBases:

Los nodos DataBase emplean una arquitectura peer to peer (p2p), comunicándose entre ellos, sin necesidad de un líder que controle el flujo de datos.

Cada DataBase expone una API REST para responder a las solicitudes que reciben desde los Workers o el resto de DataBases.

Cada nodo mantiene su propio almacenamiento, procesa consultas de lectura y escritura, y participa en el mecanismo de replicación distribuida para lograr la consistencia del sistema.

Procesos de los DataBases:

Los DataBase realizan 5 procesos distintos:

1. RESTs: Encargados de procesar y darle respuesta a las peticiones de los workers u otros DataBases. (son los procesos detrás de los endpoints que exponen).

Cuando un DataBase recibe una petición primeramente mira si el es capaz de realizarla (si tiene los datos necesarios guardados localmente) en caso de no poder busca un nodo que si pueda (a través de las tablas de rutas) y le realiza la petición a ese nodo.

2. Descubrimiento: Hilo que se encarga a través del DNS de Docker de saber que nodos DataBase están activos.

Mantiene una tabla con todos los nodos del sistemas (incluso los que no están activos), y a cada uno le asocia una variable health (bool que dice si está activo o no).

3. Coordinación: Para coordinarse, los DataBases necesitan saber que nodos guardan cada dato, y esto lo hacen compartiendo unas tablas (llamadas tablas de rutas), las cuales precisamente relacionan los **ids** de los datos con las **ips** de los nodos que almacenan la información de dichos datos.

El sistema busca que entodo momento los nodos tengan las tablas lo más actualizado posible, por lo que cada vez que ocurre un cambio en alguna de las tablas el nodo que realizó el cambio la comparte con el resto de nodos.

Notemos que las tablas de ruta solo cambian en dos situaciones específicas: - Cuando se agrega un nuevo dato al sistema. - Cuando falla algún nodo.

Para el caso en que se agrega un nuevo dato al sistema el nodo DataBase que recibió la petición simplemente guarda el dato e inicia un proceso de replicación del dato. Una vez terminadas las replicaciones actualiza sus tablas de rutas y se las comparte a los demás nodos, para que la actualicen.

Para el caso en que falla un nodo (o varios nodos):

Primero definamos como **"dueño de un dato"** al nodo de la red que está activo y es el de menor ip que almacena el dato.

Cada nodo DataBase mantiene en ejecución un hilo que con ayuda de la lista de ips activos que proporciona el descubrimiento y con las tablas de rutas, se encarga de verificar si un dato no está replicado la cantidad de veces necesarias en nodos activos, y en caso de que un dato no lo esté solo el dueño de dicho dato iniciará un proceso de replicación, actualizará sus tablas de rutas y las compartirá con el resto de nodos.

4. Replicacion: El proceso de replicación está directamente vinculado al proceso de coordinación ya que solo lo inicia dicho proceso.

Este proceso recibe un dato y emplea las tablas de rutas y en los nodos activos para copiar el dato a nodos activos que no lo tengan mientras sea posible o hasta que alcance la cantidad de replicaciones necesarias para garantizar la tolerancias a fallos del sistema.

A medida que va copiando los datos actualiza las tablas de rutas para que luego el proceso de coordinación se encargue de compartirlas.

5. Consistencia: La cosistencia se basa en tratar de que todos los nodos que almacenan un dato tengan la misma versión del mismo.

Para ello cada nodo DataBase tiene un hilo que verifica cada cierto tiempo si es el dueño de un dato, y en caso de serlo verifica las versiones del dato en los nodos que lo tienen replicado, si todas las versiones son las mismas no hay problema, pero en caso de haber diferencias (solo pueden haber en los modelos ya que los trainings y los datasets no se cambian nunca) realiza un "merge".

El "merge" de los modelos se hace teniendo en cuenta la información de los batches por los que se quedaron entrenando y por los que se quedaron prediciendo y prioriza siempre la información del modelo que se quedó por el mayor batch; ya que los entrenamientos y las predicciones se realizan secuencialmente sobre los batches y todos los modelos del mismo tipo se inicializan usando la misma semilla.

Una vez hecho el "merge" le enváa el nuevo dato al resto de los que tenian las réplicas.

Tolerancia a fallas de los DataBases:

Los DataBases presentan una tolerancia 2(por defecto) a fallos o desconexiones temporales de nodos, pero si se aumenta el factor de replicación también aumentaría el factor de tolerancia a fallos.

Para los DataBases una partición de la red no representa ningún problema ya que sería igual a que fallen muchos nodos a la vez, es decir siempre que no fallen todos los nodos que tienen un dato guardado el sistema seguirá funcionando de manera correcta.

Al desconectarse un nodo temporalmente, si en el tiempo que se desconectó, el líder de los datos que almacenaba el nodo caído se da cuenta inicia automaticamente el proceso de replicación y al volver nuevamente el nodo habrán más nodos con el dato de los que había antes de la caída del nodo, lo cual no es malo ya que a partir de ese momento dichos datos tendrán mas tolerancia a fallos, para si en el futuro el nodo (u otro nodo) presenta fallos ya no habria que replicar nada.

En caso de que hubiera un cambio en los datos mientras el nodo no estaba disponible, al volver a unirse a la red el proceso de consistencia eventualmente se dará cuenta y el líder hara el merge de los datos y luego la replicación del mismo, con lo cual se lograría la consistencia nuevamente. Con esto se cubriría tambien el caso de que ocurra una partición de la red ya que es análogo a que muchos nodos se desconecten temporalmente a la vez.

Cuando un nodo se agrega nuevo a la red, eventualmente le tocará replicar un dato o le tocará agregar un nuevo dato al sistema, pero mientras tanto sirve como router para las peticiones de los Workers, por lo que tampoco presenta ningún problema para el sistema.

Comunicación

La comunicación entre todos los componentes del sistema (workers, nodos DataBase y clientes) se realiza exclusivamente mediante peticiones HTTP.

Esto debido a que cada nodo (excepto los clientes) expone una API REST que permite intercambiar datos y coordinar las operaciones distribuidas sin necesidad de mantener conexiones persistentes. Y tambien gracias a que todos los procesos del sistema son independientes y no necesitan mantener una conexión.

Gracias a este enfoque, los nodos se mantienen desacoplados, pueden operar de forma independiente y es posible reemplazarlos, escalarlos o reiniciarlos sin afectar el funcionamiento global del sistema.

Nombrado y Localización

Para que cada nodo del sistema pueda localizar a otros nodos activos dentro de la red, se utilizan dos mecanismos de descubrimiento complementarios. Esto garantiza que, si uno falla, el otro pueda actuar como respaldo, manteniendo la comunicación y la operatividad del sistema.

Domain Name Service (DNS)

El sistema se ejecuta sobre una red de Docker utilizando Docker Swarm, que proporciona un DNS interno capaz de resolver automáticamente los nombres de servicio de cada nodo.

Gracias a este DNS:

- Los nodos no necesitan conocer direcciones IP específicas.
- Un worker puede comunicarse con los nodos DataBase simplemente usando su dominio o nombre de servicio.
- Docker gestiona la resolución y el balanceo entre instancias activas.
- Los fallos de nodos individuales quedan aislados del resto del sistema.
- Este mecanismo es la forma principal de descubrimiento dentro de la red.

IP Cache

Como mecanismo de respaldo, cada nodo mantiene una caché local de direcciones IP correspondientes a otros nodos conocidos. Está técnica funciona incluso si la resolución por DNS falla.

El funcionamiento es el siguiente:

IPs fijas al iniciar el nodo:

Cuando un nodo se levanta, carga en su caché un conjunto de IPs fijas preconfiguradas. Esto le permite tener acceso inmediato a nodos potenciales, incluso si el DNS presenta fallos desde el inicio.

Actualización dinámica:

A medida que el nodo interactúa con otros, añade nuevas IPs a la caché cuando detecta nodos que responden correctamente.

Uso como respaldo:

Si el DNS no funciona o la red presenta inconsistencias, el nodo reutiliza las IPs guardadas en la caché para intentar conectarse a otros nodos activos.

Depuración automática:

Si una IP en la caché deja de ser válida, se descarta al detectar fallos consecutivos.

Gracias a está combinación, el sistema puede arrancar, operar y recuperarse incluso en momentos donde la infraestructura de red esté parcialmente degradada.

Distribución de servicios en ambas redes de docker

Dicha arquitectura permite que el sistema funcione en una red de docker siempre que exista al menos un nodo de cada tipo.