Técnicas para Big Data

Clase 09 - Bases de Datos Distribuidas

Hasta ahora

Hemos visto técnicas para cuando los datos nos caben en un solo servidor

¿Pero qué pasa cuando los datos no nos caben en un solo computador?

En esta clase

- Bases de datos distribuidas
- Teorema CAP
- Map - Reduce y Hadoop

Bases de Datos Distribuidas

¿Por qué BDs distribuidas?

Una base de datos distribuida puede ser útil en varios contextos

- En caso de muchos datos, utilizar más de un disco duro
- Para buscar más *performance*, podemos paralelizar operaciones
- Podemos replicar datos, para no perder información si se cae un servidor
- ...

Bases de Datos Distribuidas

Ya sabemos que un DBMS es una pieza de software compleja, ya que debe ser capaz de:

- Guardar información de forma consistente
- Proveer acceso concurrente
- Hacer uso óptimo del buffer para responder consultas
- Optimizar las consultas que recibe (plan lógico - plan físico)
- ...

Bases de Datos Distribuidas

Las técnicas que hemos visto hasta ahora, ¿se pueden aplicar en un entorno distribuido?

En general, las técnicas para responder consultas de forma eficiente, proveer acceso concurrente y mantener consistencia **deben ser adaptadas**

Bases de Datos Distribuidas

Ejemplo - Join Distribuido

Supongamos que tenemos dos relaciones, A y B, que están distribuidas a lo largo de varios servidores

Si queremos hacer un *join* por un atributo que **no es la llave primaria**, una técnica posible es hacer un *hash join* por bloques

Esto es, descomponer $A \bowtie B$ en joins más pequeños, utilizando los servidores como *buckets*, donde se van a procesar según la partición dada por la función de *hash*

Bases de Datos Distribuidas

Arquitecturas

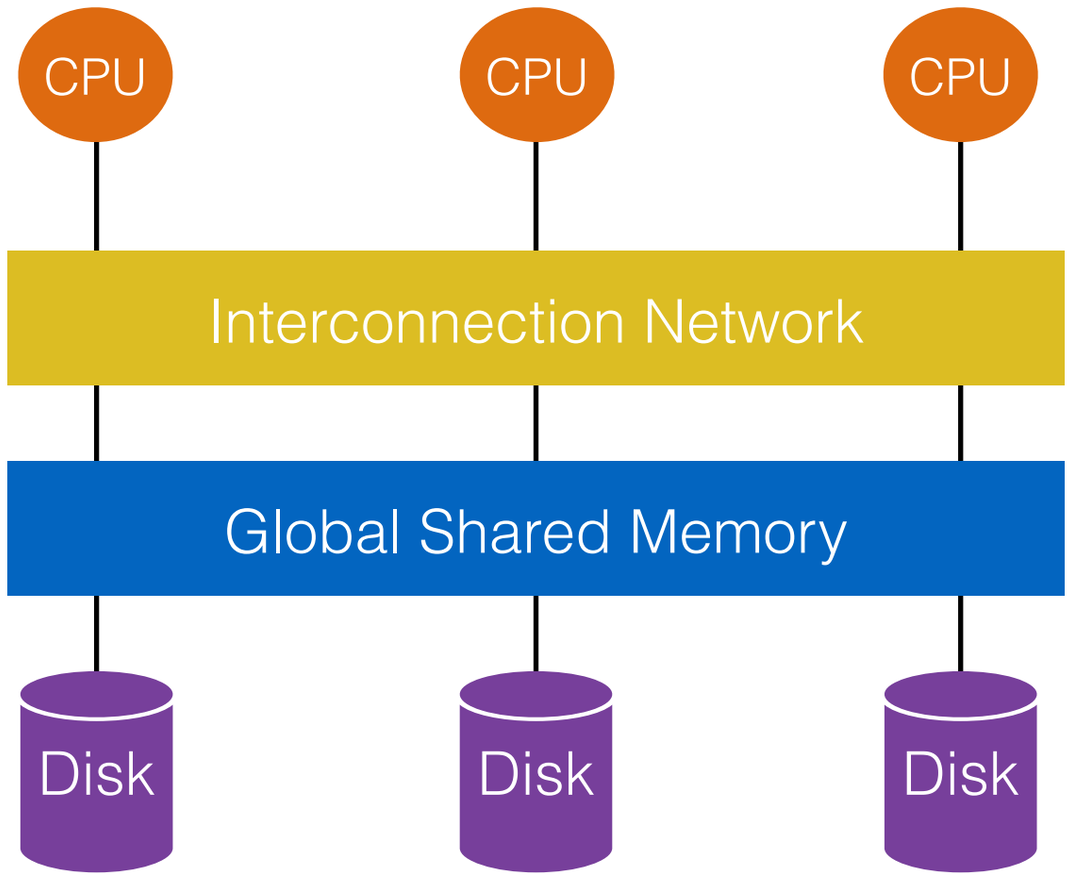
En el contexto de bases de datos distribuidas, tenemos 3 arquitecturas principales:

- Shared Memory
- Shared Disk
- Shared Nothing

Bases de Datos Distribuidas

Arquitectura Shared Memory

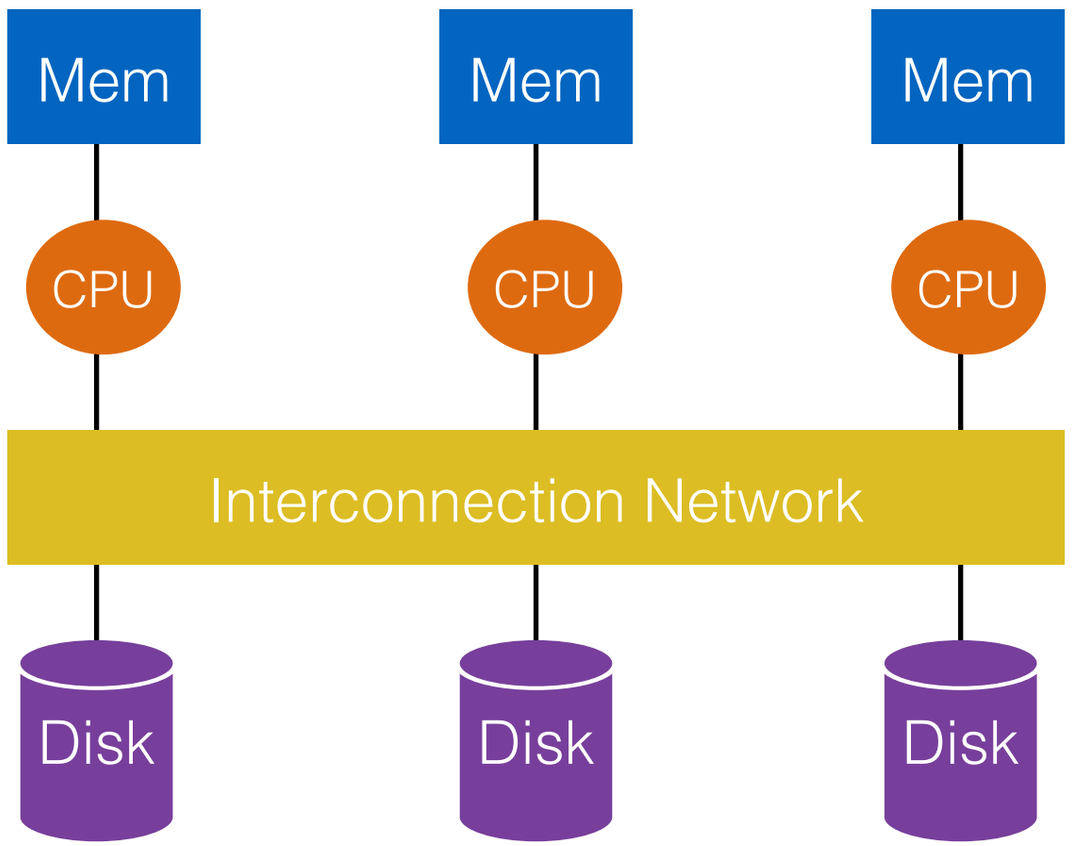
En la arquitectura **Shared Memory** varias CPU se conectan a una *interconnection network*, y pueden acceder a una **Memoria Compartida**



Bases de Datos Distribuidas

Arquitectura Shared Memory

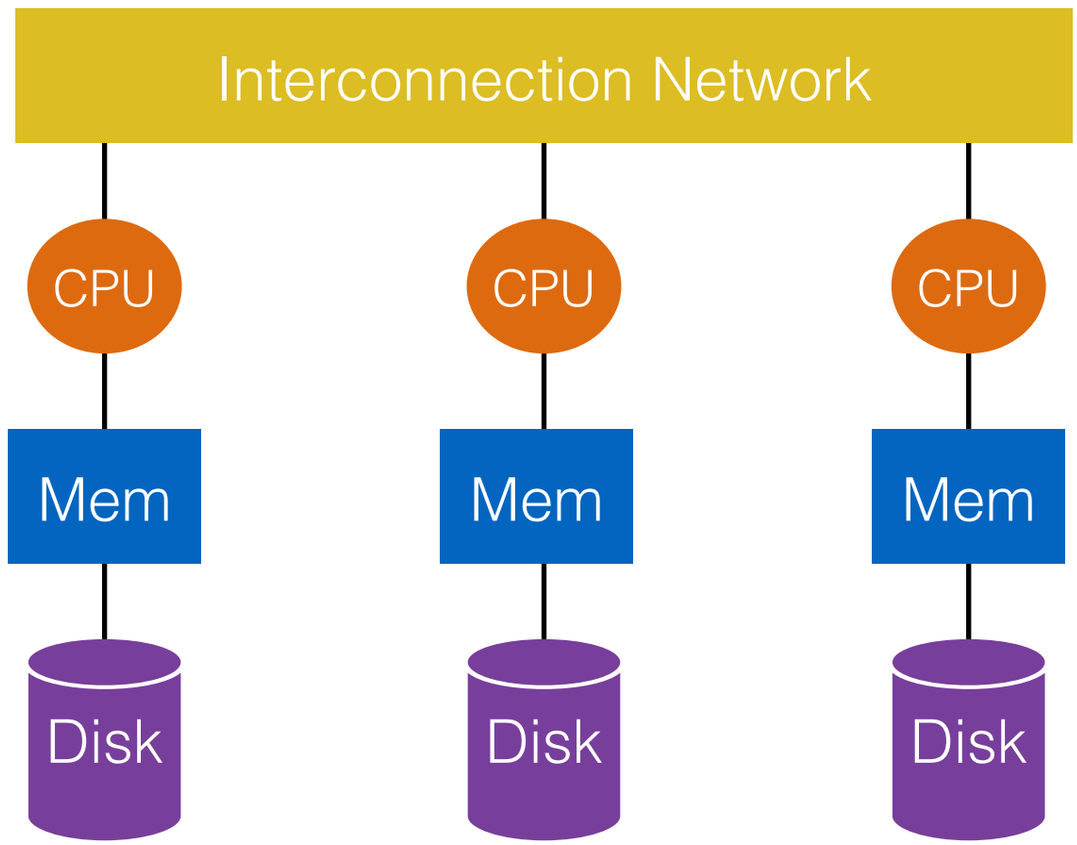
En la arquitectura **Shared Disk** cada CPU tiene su propia memoria privada, pero la *interconnection network* sirve para compartir todos los discos



Bases de Datos Distribuidas

Arquitectura Shared Nothing

La arquitectura **Shared Nothing** cada CPU tiene una memoria y un disco local, pero las CPU no comparten el área de *storage*; toda la comunicación se hace a través de la red



Sharding

La forma más común de distribuir una base de datos relacional es hacer **Sharding**

Sharding es el proceso de dividir una tabla en varias partes, que se distribuyen a través de varios servidores

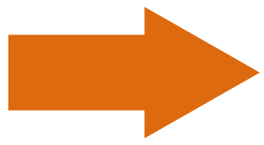
Existen dos tipos de Sharding, vertical y horizontal

Sharding

Sharding horizontal

Cuando uno habla de Sharding, en general se refiere a **Sharding Horizontal**; esto es, dividir la tabla en varios conjuntos de tuplas, y enviar cada conjunto a un servidor

R.a	R.b	R.c
1	1	1
2	2	2
3	3	3
4	4	4



Servidor 1

R.a	R.b	R.c
1	1	1
2	2	2

Servidor 2

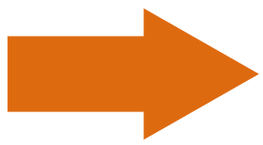
R.a	R.b	R.c
3	3	3
4	4	4

Sharding

Sharding vertical

Por otra parte, **Sharding Vertical** es dividir la tabla en columnas, y guardar cada una en un servidor

R.a	R.b	R.c
1	1	1
2	2	2
3	3	3
4	4	4



Servidor 1

R.a	R.b
1	1
2	2
3	3
4	4

Servidor 2

R.c
1
2
3
4

Teorema CAP

Limitaciones de una BD distribuida

Utilizar una BD distribuida puede traer un número considerable de problemas:

- ¿Cómo distribuimos los updates a todos los servidores?
- ¿Qué pasa si la comunicación entre servidores se cae?
- ¿Qué pasa si un servidor se cae?
- ...

El Teorema CAP

Uno de los resultados más interesantes en el contexto de BDs distribuidas es el Teorema CAP

El teorema postula que un sistema de almacenamiento de datos distribuido no puede proveer más de dos de las siguientes 3 características:

- Consistencia (*Consistency*)
- Disponibilidad (*Availability*)
- Tolerancia a las particiones (*Partition tolerance)*

El Teorema CAP

Consistencia. Cada lectura al sistema recibe la actualización más reciente, o bien un error

Availability. Cada request recibe una respuesta (que no sea error), sin la garantía de que se contenga la actualización más reciente

Tolerancia a las particiones. El sistema sigue operando a pesar de que haya un número arbitrario de mensajes entre los servidores que se pierden (i.e. se cae la conexión)

El Teorema CAP

Así, en caso de una falla en la red, tenemos que escoger entre:

- Cancelar las operaciones, y por lo tanto no tendremos disponibilidad pero aseguraremos consistencia
- Seguir con las operaciones y proveer disponibilidad a cambio de transar en consistencia

Ojo. La noción de consistencia propuesta por los autores difiere un poco de la definición en el contexto ACID

El Teorema CAP

Importante. A veces este teorema se presta para confusiones, principalmente enunciando que uno tiene que transar siempre una de las tres propiedades

En realidad, la elección entre consistencia y disponibilidad la debemos hacer solamente cuando tenemos una partición en la red; en otro contexto (i.e. con la red funcionando) no hay que hacer ninguna elección

Map-Reduce y Hadoop

Map-Reduce

- Técnica eficiente para computación paralela
- Paradigma de programación - mezcla de datos con controlador
- No está orientado a funcionar como un motor de datos, sino como una herramienta para hacer consultas
- Cada servidor no conoce lo que tiene el resto y no podemos darnos el lujo de comunicar todos los datos

Map-Reduce

- Map: recibe datos y genera pares key - values
- Reduce: recibe pares con el mismo key y los agrega

Map-Reduce

- Map: recibe datos y genera pares key - values
- Reduce: recibe pares con el mismo key y los agrega

Ejemplo: ver palabra más utilizado en un texto T

Map-Reduce

Ejemplo

Map:

- Recibe un pedazo de texto
- Por cada palabra, emite el par `(palabra, número de ocurrencias)`

Reduce:

- Cada reduce recibe todos los pares asociados a la misma palabra
- Junta todos estos pares y suma las ocurrencias

Map-Reduce

Arquitectura

Mappers:

- Nodos encargados de hacer Map
- Reciben parte del documento y lo envían a los reducers

Reducers:

- Nodos encargados de hacer Reduce
- Reciben los Map y los agregan; el output es la unión de cada Reducer

Ojo. El *output* de una operación Reduce puede ser el *input* de otro Map

Map-Reduce

No es un descubrimiento nuevo, pero recientemente se ha visto calzar perfectamente con las necesidades de las grandes fuentes de datos

Es la arquitectura más importante en sistemas que reciben grandes bases de datos

Hadoop: La implementación de Google de Map -Reduce, presente en muchos sistemas con computación distribuida

Map-Reduce

Ejemplo: Join

¿Cómo hago un join con Map Reduce?

- Modelo: un archivo con el nombre de la tabla y sus tuplas

Solución completa **en vivo**, la idea es:

- Map: Agrupo por el atributo que hace el join
- Reduce: Hago el producto cruz para las tablas distintas

Técnicas distribuidas en la práctica

Técnicas Distribuidas

Ahora vamos a ver un ejemplo en vivo de:

- Cómo distribuir una BD PostgreSQL
- Cómo hacer un programa con Hadoop
- Además, vamos a ver un poco de cómo manejar servidores

Material Adicional

Algunos recursos de interés:

- Uber y la elección de motor SQL (https://eng.uber.com/postgres-to-mysql-migration/)
- Demostración ilustrada del teorema CAP (https://mwhittaker.github.io/blog/an_illustrated_proof_of_the_cap_theorem/)
- Netflix Simian Army (https://netflixtechblog.com/the-netflix-simian-army-16e57fbab116)
- Tutorial de distribución de Postgres (https://dev.to/aurelmegn/setting-up-distributed-database-architecture-with-postgresql-261)
- Recurso oficial de PSQL sobre distribución de datos (https://www.postgresql.org/docs/9.1/ddl-partitioning.html)

Técnicas para Big Data

Clase 09 - Bases de Datos Distribuidas