DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Bases de datos distribuidas

© Fernando Berzal, berzal@acm.org

Acceso a los datos

- Bases de datos relacionales: SQL
- O/R Mapping
- Bases de datos distribuidas
- Bases de datos NoSQL
- Bases de datos multidimensionales: Data Warehousing

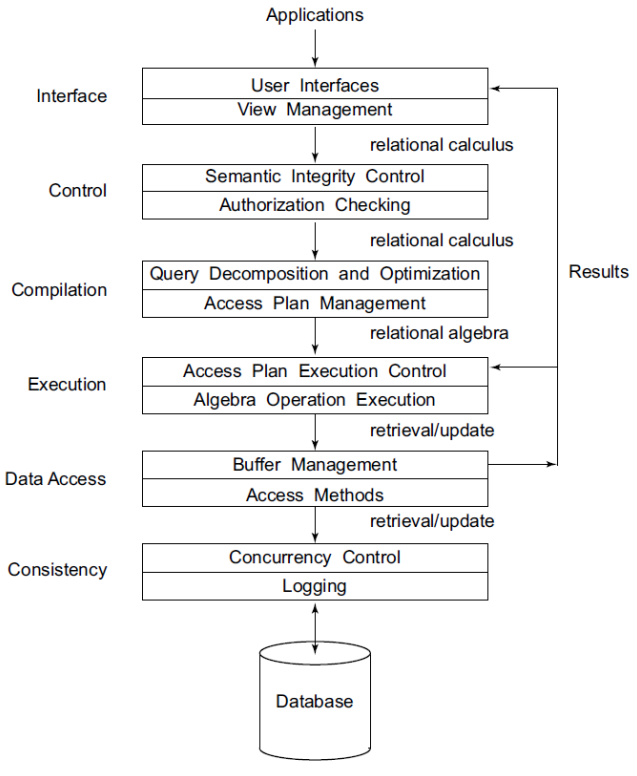
Bases de datos distribuidas

- Arquitectura de un DBMS
  - Sistemas C/S
  - Sistemas P2P
  - Sistemas MDBS
- Diseño de bases de datos distribuidas
  - Fragmentación
  - Asignación de recursos
- Replicación de datos

Arquitectura de un DBMS

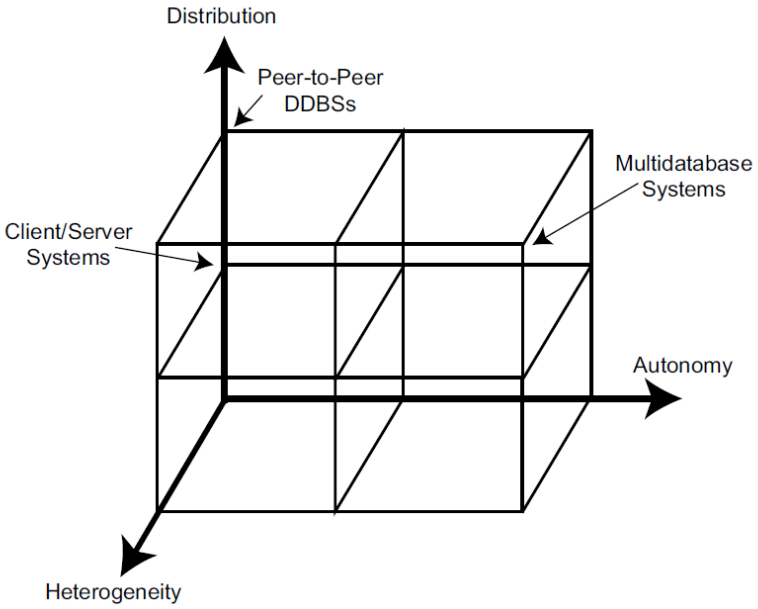
Capas funcionales

DBMS centralizado



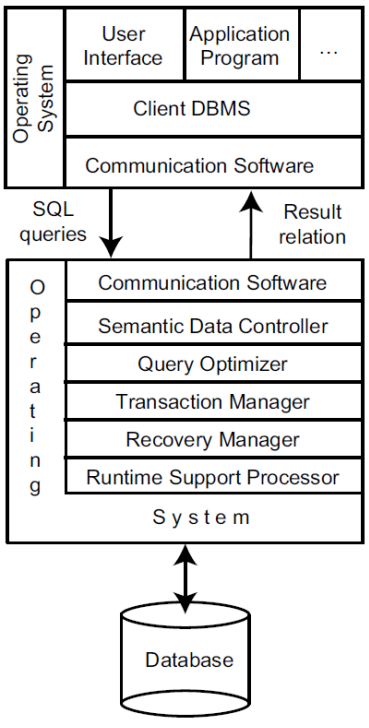
Arquitectura de un DBMS

Alternativas de implementación de un DBMS distribuido



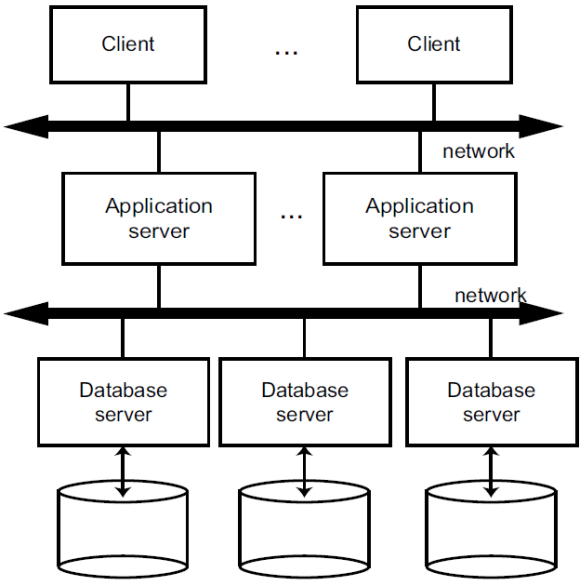
Arquitectura de un DBMS

Sistemas cliente/servidor



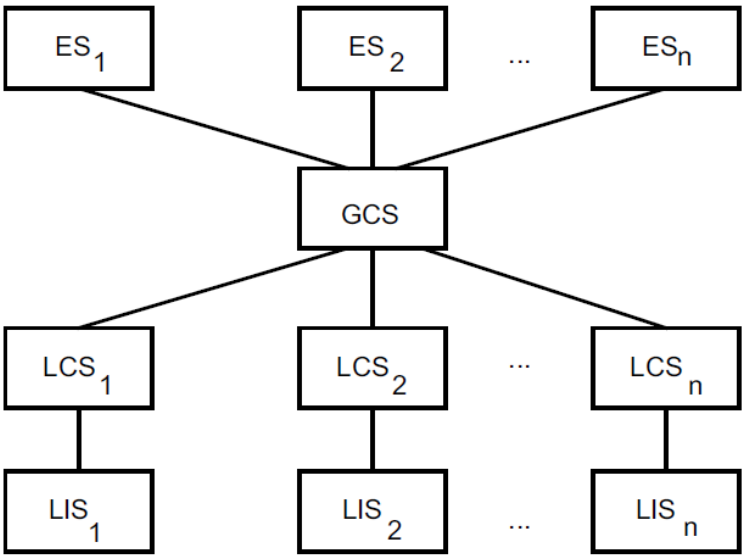
Arquitectura de un DBMS

Sistemas cliente/servidor
con múltiples servidores



Arquitectura de un DBMS

Bases de datos distribuidas
Extensión del modelo ANSI/SPARC



ES
External Schema

GCS
Global Conceptual Schema

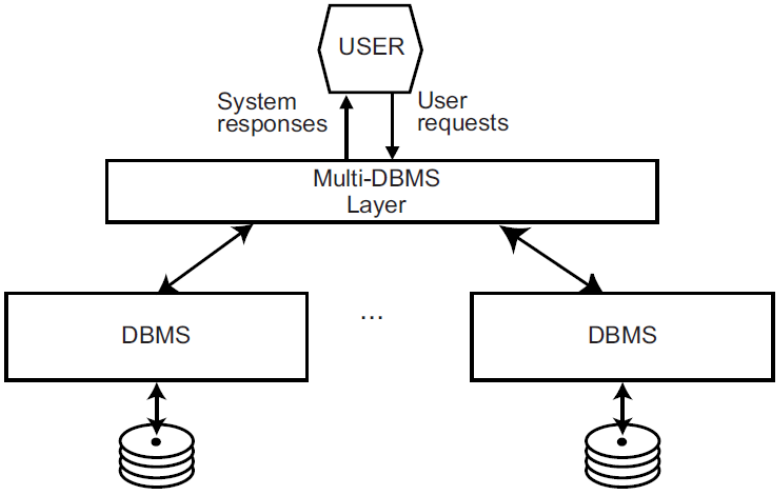
LCS
Local Conceptual Schema

LIS
Local Internal Schema

Arquitectura de un DBMS

Sistemas MDBS

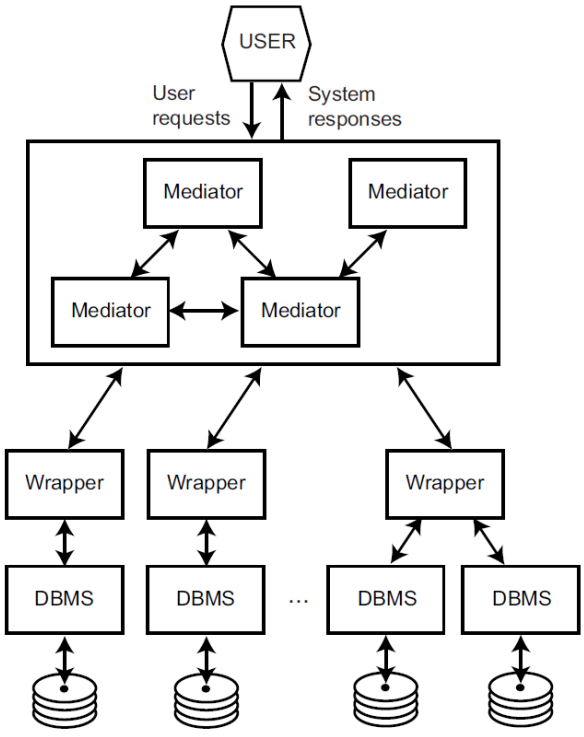
Multidatabase Systems



Arquitectura de un DBMS

Sistemas MDBS

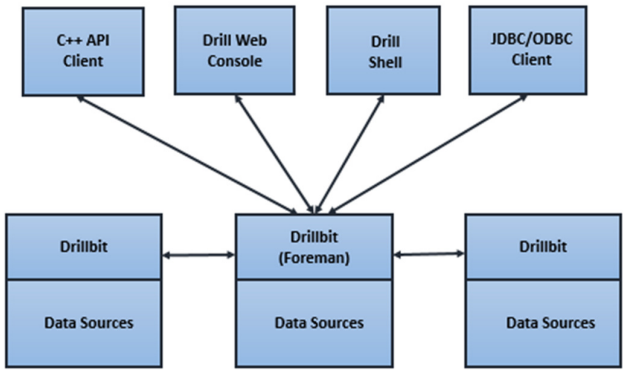
Mediators & wrappers



Arquitectura de un DBMS

Ejemplo: Apache Drill

https://drill.apache.org/

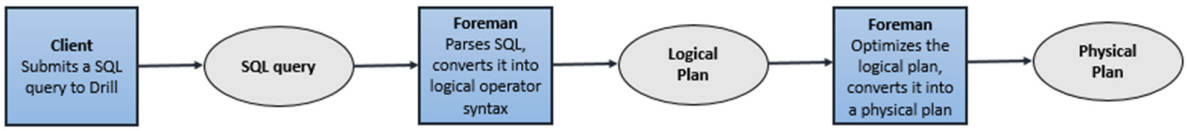


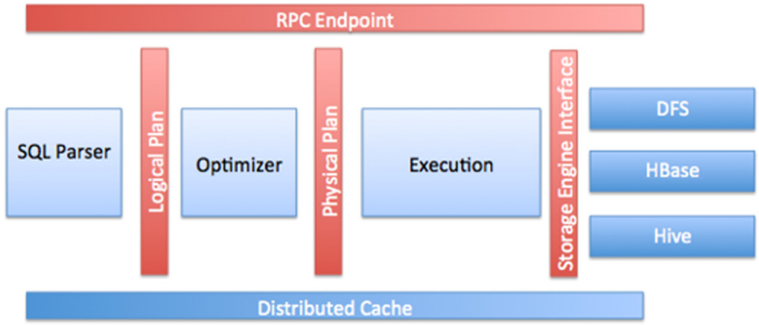
NOTA: Inspirado en Dremel (Google, VLDB'2010), base de Google BigQuery [IaaS]: https://cloud.google.com/bigquery/

Arquitectura de un DBMS

Ejemplo: Apache Drill

https://drill.apache.org/

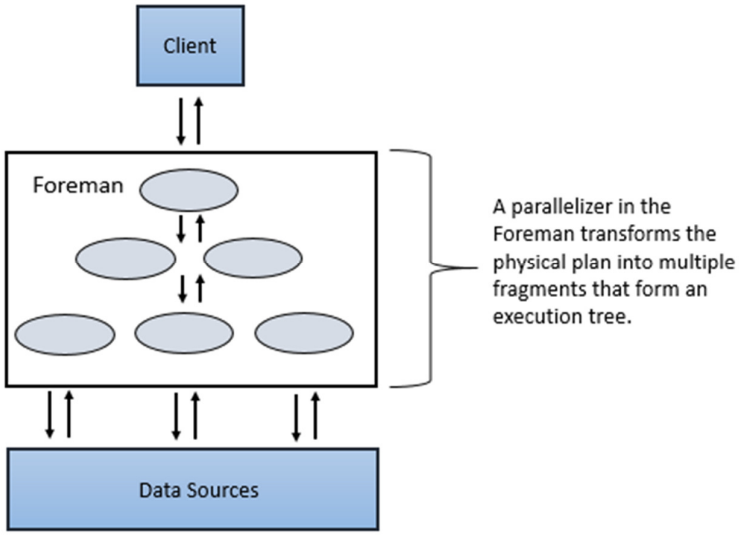




Arquitectura de un DBMS

Ejemplo: Apache Drill

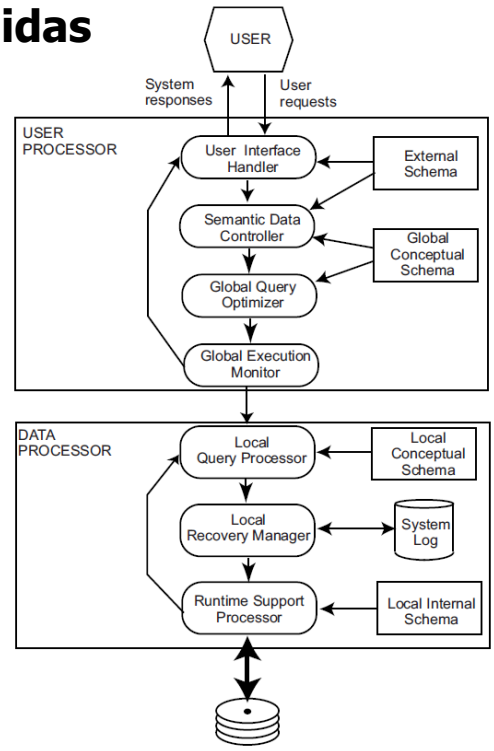
https://drill.apache.org/



Arquitectura de un DBMS

Bases de datos distribuidas

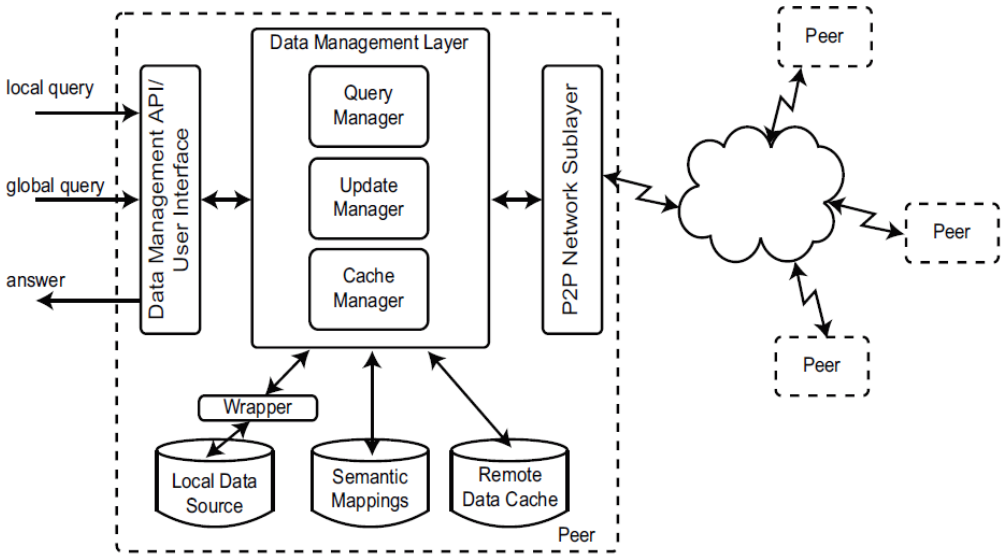
DBMS distribuido



Arquitectura de un DBMS

Sistemas P2P [Peer-to-peer]

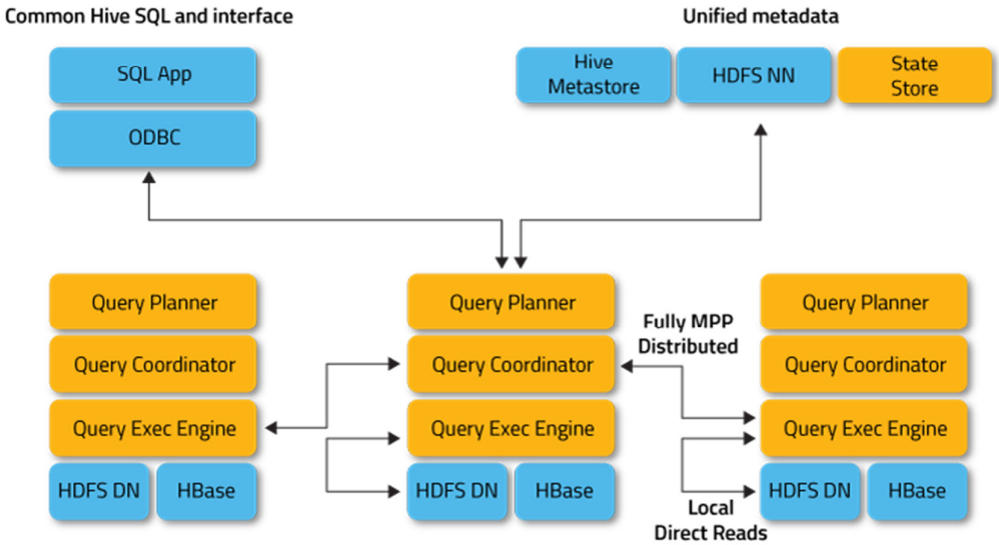
Arquitectura



Arquitectura de un DBMS

Ejemplo: Impala (Cloudera)

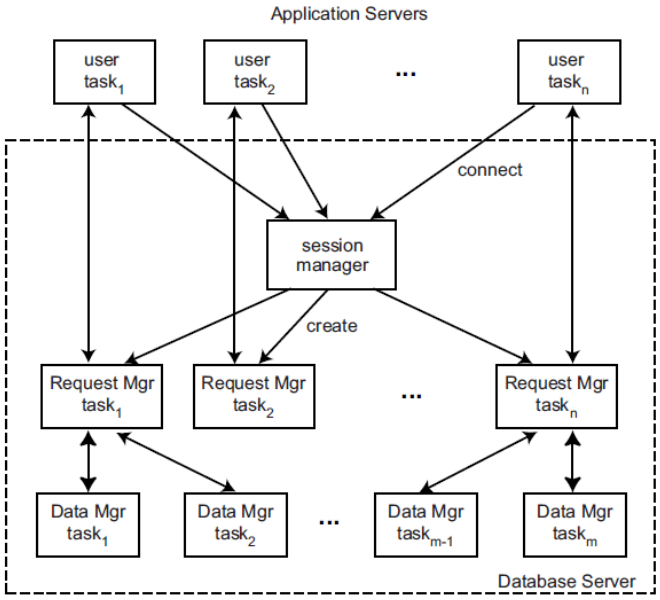
http://impala.io/

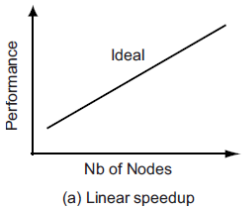


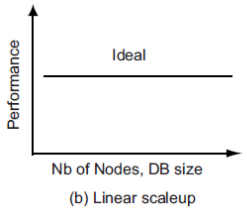
Arquitectura de un DBMS

Parallel DBMS

e.g. MPP DBMS [Massively Parallel Processing DBMS]





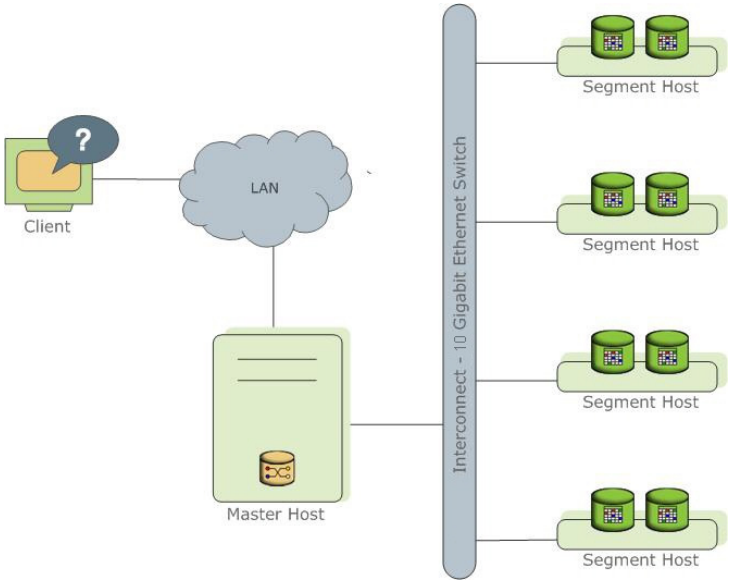


Arquitectura de un DBMS

Ejemplo: Greenplum (Pivotal Software)

http://greenplum.org/

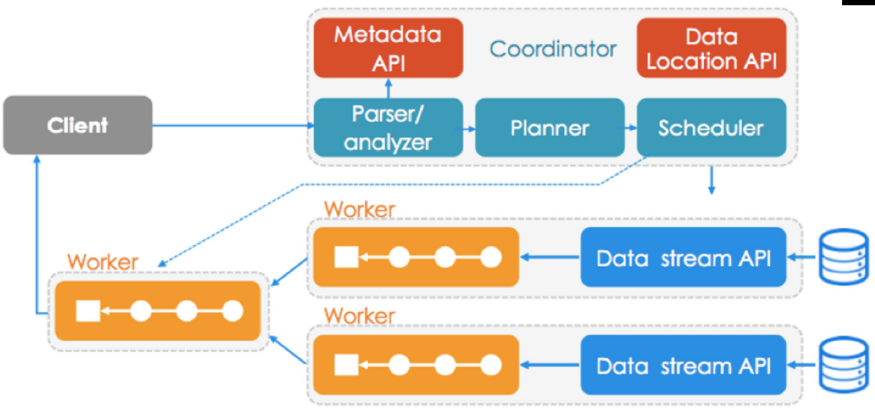
GREENPLUM DATABASE



Arquitectura de un DBMS

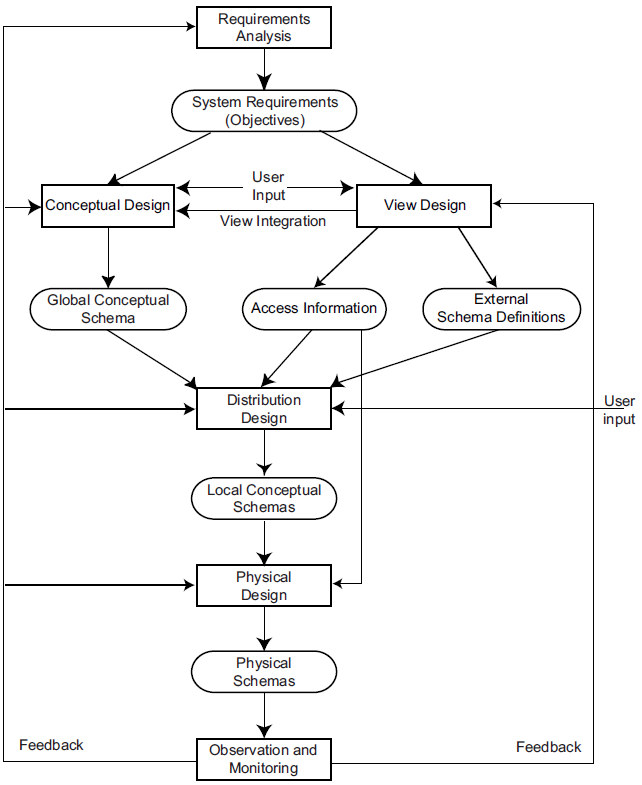
Ejemplo: PrestoDB (Facebook)

https://prestodb.io/



Usuarios:
Facebook (300PB Hive), Netflix (25PB Amazon S3), AirBnB (1.5PB Hive), Groupon (HBase)…

Diseño



Diseño

Fragmentación

¿Por qué fragmentar los datos?

- Localidad de acceso
  (las vistas utilizadas por las aplicaciones suelen ser subconjuntos de las relaciones globales).
- Distribución de los datos
  (minimización del volumen de acceso a datos remotos).
- Rendimiento
  (ejecución en paralelo de consultas y transacciones).

Diseño

Fragmentación

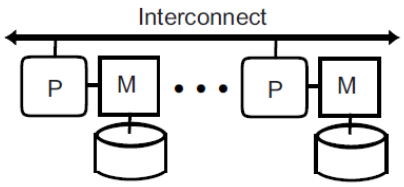
Estrategias de fragmentación de los datos:

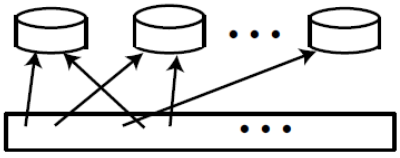
- Fragmentación horizontal (selección / tuplas)
- Fragmentación vertical (proyección / columnas)
- Fragmentación anidada (híbrida)

Diseño

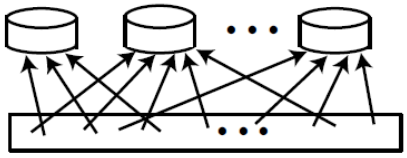
Particionamiento

Fragmentación horizontal en sistemas distribuidos

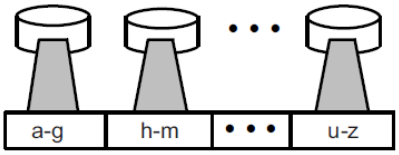




(a) Round-Robin



(b) Hashing

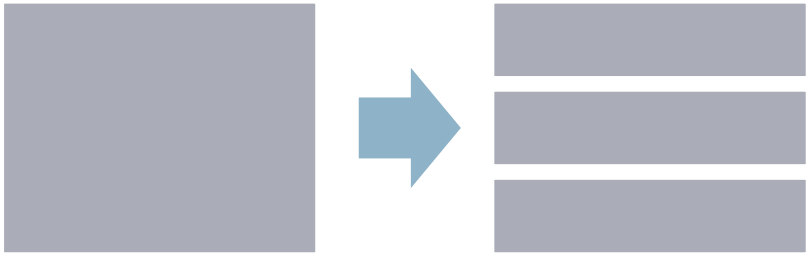


(c) Range

Diseño

Fragmentación horizontal

- Fragmentación horizontal primaria (predicados definidos sobre la propia relación).
- Fragmentación horizontal secundaria (predicados definidos sobre otras relaciones).



Diseño

Fragmentación horizontal

Información necesaria (cualitativa y cuantitativa):

Base de datos: Esquema conceptual global (GCS).

- Enlaces entre relaciones (claves externas).
- Cardinalidad de cada relación.

Aplicaciones:

- Predicados utilizados en las consultas (regla 80/20: 20% de las consultas, 80% de los accesos).
- Selectividad de los términos de las consultas (número de tuplas a las que se accede al utilizar un término).
- Frecuencias de acceso.

Diseño

Fragmentación horizontal primaria

$\mathbf{R_i = \sigma_{Fi}(R)} \qquad \mathbf{1 \leq i \leq w}$

Propiedades deseables:

- Predicados completos (misma probabilidad de acceso a cualquier tupla de cualquier fragmento)
  ➔ Balanceado de carga.
- Predicados minimales (dados dos fragmentos f_i y f_j, al menos una aplicación debería acceder a ellos de forma diferente): $acceso(F_i)/card(R_i) \neq acceso(F_j)/card(R_j)$

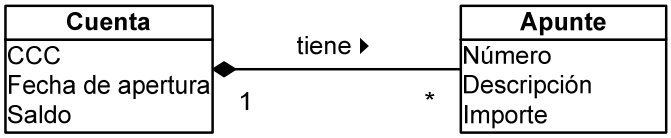
Diseño

Fragmentación horizontal derivada

$R_i = R \ltimes \sigma_{Fi}(S) \qquad 1 \leq i \leq w$

La fragmentación se define sobre R a partir de su "propietaria" S mediante una equi-reunión [equi-join].

Ejemplo: Entidades débiles



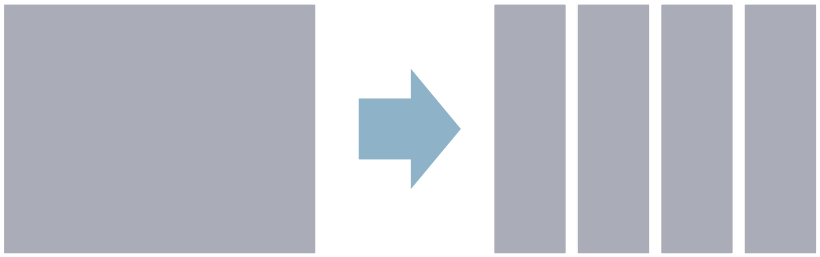
Diseño

Fragmentación vertical

OBJETIVO:

Dividir una relación en relaciones más pequeñas de forma que cada aplicación trabaje sólo con un fragmento.

¡Peligro! La reconstrucción de los datos requiere el uso de reuniones (operaciones costosas, más si son distribuidas).

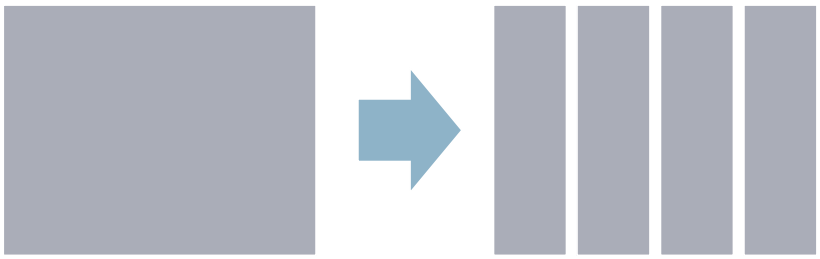


Diseño

Fragmentación vertical

También en bases de datos centralizadas:

- Disminución del número de accesos a páginas de disco (menor tamaño de las relaciones).
- Uso de cachés (memoria más rápida para fragmentos a los que se accede con mayor frecuencia).



Diseño

Fragmentación vertical

Criterios heurísticos:

- Agrupamiento [grouping]: Inicialmente, cada atributo a un fragmento, que se va combinando con otros hasta que se cumplan las condiciones deseadas
  ➔ Fragmentos con solapamiento.

- División [splitting]: Se decide la fragmentación en función del acceso de las aplicaciones a los datos
  ➔ Fragmentos no solapados.

  NOTA: La clave primaria, por razones obvias, sí se replica.

Diseño

Fragmentación vertical

Información necesaria (cualitativa y cuantitativa):

Aplicaciones (para colocar en el mismo fragmento los atributos a los que se accede conjuntamente):

- Predicados utilizados en las consultas (regla 80/20: 20% de las consultas, 80% de los accesos).
- Frecuencias de acceso de las consultas a los atributos individuales: use(q_i,A_j).
- Afinidad entre los atributos (a partir del número de accesos conjuntos a los atributos y la frecuencia de cada consulta): aff(A_i,A_j).

Diseño

Distribución de los datos [data allocation]

PROBLEMA DE OPTIMIZACIÓN

Dados un conjunto de fragmentos F
y un sistema distribuido con un conjunto de nodos N
sobre el que debe funcionar un conjunto de aplicaciones Q,
encontrar la distribución "óptima" de F sobre N.

Criterios de optimalidad:

- Coste mínimo (almacenamiento & comunicación).
- Rendimiento (tiempo de respuesta o throughput).

Diseño

Distribución de los datos [data allocation]
Información necesaria

Base de datos:
- Selectividad (de fragmentos con respecto a consultas).
- Tamaño de los fragmentos (MB).

Aplicaciones:
- Accesos de lectura y accesos de actualización.

Sistema de almacenamiento de datos:
- Capacidad de almacenamiento de cada nodo.
- Capacidad de procesamiento de cada nodo.
- Coste de comunicación entre nodos.

Diseño

Distribución de los datos [data allocation]
Problema de optimización

DAP [Database Allocation Problem]
Minimizar el coste total
sujeto a…
restricciones de tiempo de respuesta
restricciones de almacenamiento
restricciones de procesamiento

Diseño

Distribución de los datos [data allocation]

$$\min \left[\sum_{i=1}^{m} \left(\sum_{j|S_j \in I} x_j u_j c'_{ij} + t_j \min_{j|S_j \in I} c_{ij} \right) + \sum_{j|S_j \in I} x_j d_j \right]$$

Modelo (más que) simplificado:

- Coste de transmisión de las actualizaciones.
- Coste de ejecución de las consultas.
- Coste de almacenamiento de las réplicas de un fragmento.

Incluso así, el problema es NP-completo

➔ Soluciones heurísticas subóptimas.

Replicación de datos

Objetivos

- Disponibilidad
  (datos accesibles aunque se produzcan fallos)
- Rendimiento
  (mover los datos cerca de su punto de acceso)
- Escalabilidad
  (tiempo de respuesta de las consultas)
- Requisitos de las aplicaciones (p.ej. legales)

Replicación de datos

Decisiones de diseño

¿Dónde se permiten las actualizaciones?

- Actualizaciones centralizadas (copia "maestra")
- Actualizaciones distribuidas

¿Cómo se propagan las actualizaciones?

- De forma síncrona: "Eager update propagation" (en el contexto de la transacción actual).
- De forma asíncrona: "Lazy update propagation" (la transacción termina sin esperar la propagación).

Replicación de datos

Actualizaciones centralizadas "eager"

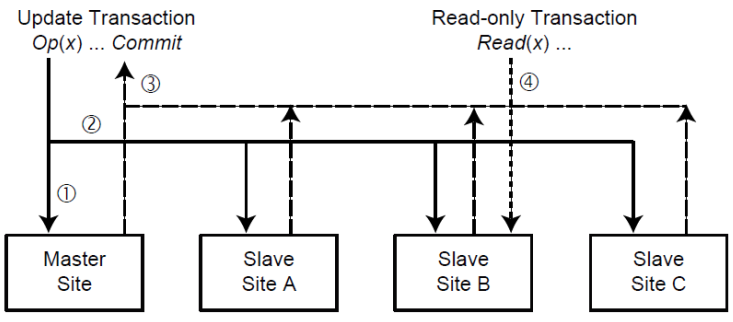


Fig. 13.1 Eager Single Master Replication Protocol Actions. (1) A *Write* is applied on the master copy; (2) *Write* is then propagated to the other replicas; (3) Updates become permanent at commit time; (4) Read-only transaction's *Read* goes to any slave copy.

Replicación de datos

Actualizaciones centralizadas "lazy"

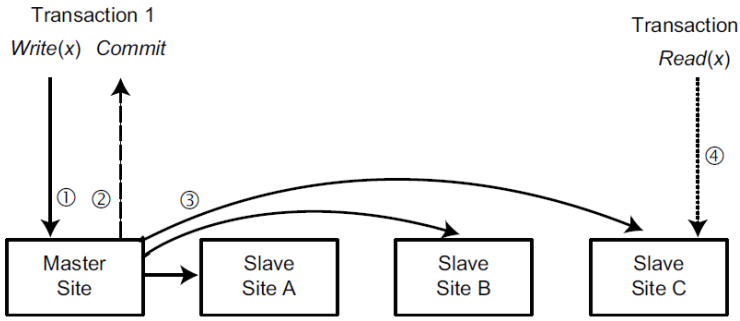


Fig. 13.4 Lazy Single Master Replication Protocol Actions. (1) Update is applied on the local replica; (2) Transaction commit makes the updates permanent at the master; (3) Update is propagated to the other replicas in refresh transactions; (4) Transaction 2 reads from local copy.

Replicación de datos

Actualizaciones distribuidas "eager"

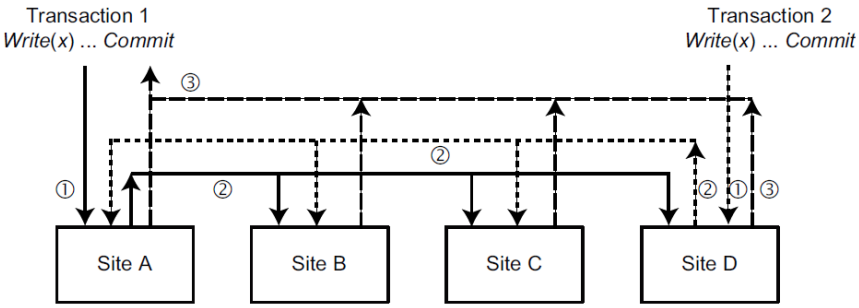


Fig. 13.3 Eager Distributed Replication Protocol Actions. (1) Two *Write* operations are applied on two local replicas of the same data item; (2) The *Write* operations are independently propagated to the other replicas; (3) Updates become permanent at commit time (shown only for Transaction 1).

Replicación de datos

Actualizaciones distribuidas "lazy"

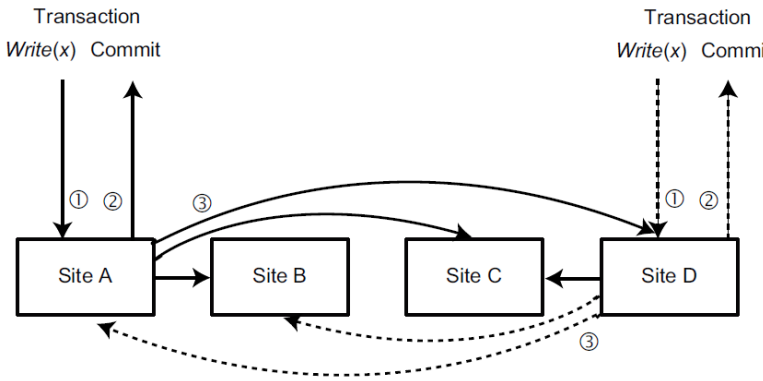


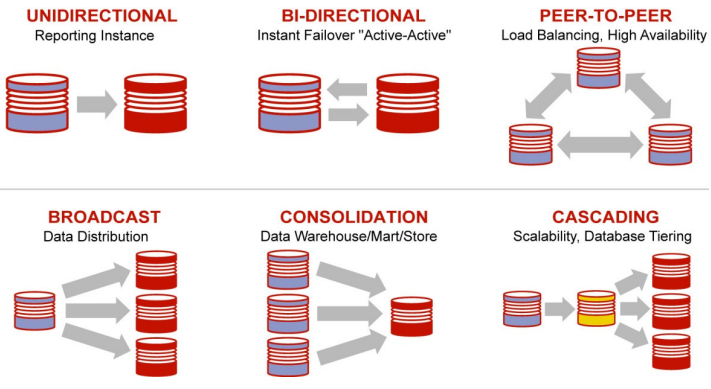
Fig. 13.6 Lazy Distributed Replication Protocol Actions. (1) Two updates are applied on two local replicas; (2) Transaction commit makes the updates permanent; (3) The updates are independently propagated to the other replicas.

Replicación de datos

Soporte en DBMSs comerciales

Oracle DB

ORACLE®
DATABASE



Oracle Streams

- Balanceado de carga con múltiples réplicas.
- Actualizaciones distribuidas.

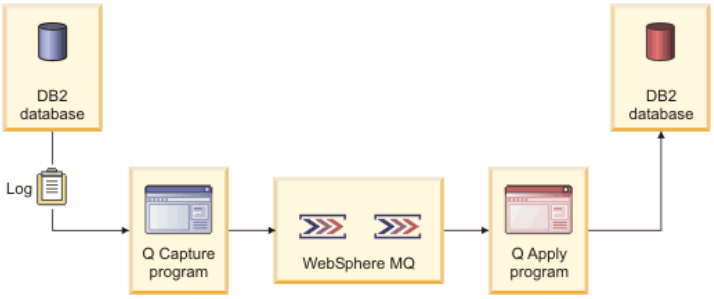
Oracle GoldenGate

- Replicación en entornos heterogéneos (p.ej. con otras bases de datos no Oracle).

Replicación de datos

Soporte en DBMSs comerciales

IBM DB2



Tres métodos diferentes de replicación:

- CDC [Change Data Capture], usando TCP/IP
- Q Replication, usando MQ Series (middleware)
- SQL Replication, usando DRDA (estándar)
  Distributed Relational Database Architecture
  https://en.wikipedia.org/wiki/DRDA

Replicación de datos

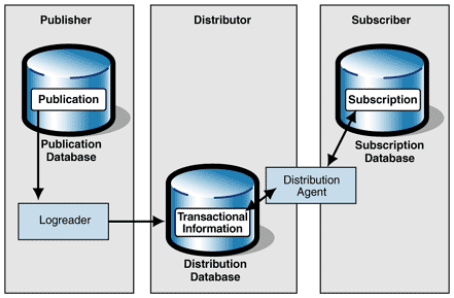
Soporte en DBMSs comerciales

Microsoft SQL Server

SQL Replication Services

Modelo "publisher/subscriber" con agentes de replicación (DBMS en el servidor o cachés en el cliente):

- Replicación de transacciones (síncrona).
- Merge (actualizaciones bidireccionales periódicas).
- Snapshot (instantánea: copia completa del estado actual de la base de datos).



Bibliografía recomendada

- M. Tamer Özsu & Patrick Valduriez:
  Principles of Distributed Database Systems.
  Springer, 3rd edition, 2011.
  ISBN 1441988335