Big Data y NoSQL

75.06 Organización de Datos – Cátedra Argerich
Facultad de Ingeniería
Universidad de Buenos Aires

ACID

1. **Atomicidad:** cada transacción es todo o nada. Si parte de una transacción falla, la transacción entera debe fallar. Un sistema atómico debe garantizar atomicidad en todas las situaciones.
2. **Consistencia:** cualquier transacción va a dejar a la base de datos desde un estado válido a otro estado válido. Todos los datos escritos en la base deben ser válidos con respecto a todas las reglas definidas.
3. **Aislamiento (isolation):** una operación no afecta a otras. Dos transacciones concurrentes sobre los mismos datos son independientes y no generan ningun tipo de error.
4. **Durabilidad:** una vez que una transacción fue commiteada, es persistente a pesar de cualquier falla que pueda ocurrir.

CAP

1. Consistente: todos los nodos tienen los mismos datos al mismo tiempo.
2. Disponibilidad: se garantiza que cada request recibe una respuesta.
3. Tolerante a particiones: sigue funcionando aunque parte se haya caido.
4. Teorema CAP: todo no se puede, elegir 2.

Mapreduce

Divide los problemas en 2 partes.

1. Primero convierte los datos en otros mediante una funcion de map.
2. Luego convierte estos datos con una funcion de reduce.
3. Permite dividir las tareas en otras más pequeñas y que puedan ser ejecutadas en paralelo en diferentes servidores.
4. Podrian ser ejecutados varios reduce sobre cada dato.
5. Por ej, contar todos los registros que tengan COD_CARRERA=10 mapeando cada padron que tenga esta carrera a count=1 y luego reduciendo la suma de todos estos count.

Relacional

1. Tablas bidimensionales, filas y columnas.
2. Estructurada, tipos de datos.
3. Relacion de datos mediante claves foráneas.
4. Basado en un modelo matemático.
5. Las relaciones se pueden combinar para generar nuevas relaciones.

6. SQL.
7. La más popular de las base de datos.
8. ACID
9. Muchas features: transacciones, vistas, triggers, stored procedures, índices, geo queries, full text search.

Ejemplos

1. MySQL
2. MSSQL
3. Oracle
4. SQLite
5. PostgreSQL.

Cuando es bueno

1. Si se sabe cómo son los datos pero no cómo se van a utilizar. SQL muy completo y flexible.

Cuando no es tan bueno

1. Cuando los datos son muy variables o con muchas jerarquias. Debido a la necesidad de definir los esquemas.

Clave-Valor

1. Mapea claves a valores, los cuales pueden ser simplemente un string, un hash, un objeto serializado o una lista (dependiendo del motor usado).
2. Sin ninguna estructura.
3. Muy rápidos para accesos por clave.
4. Simples para distribuir.
5. Muchas veces usados como cache.

Ejemplos

1. Memcached
2. Voldemort
3. Redis
4. Riak
5. Tokyo/Kyoto Cabinet

6. Google LevelDB.

Cuando es bueno

1. Cuando se quiere escalar horizontalmente o ser muy rápido.
2. Cuando los datos no estan muy relacionados.

Cuando no es tan bueno

1. Cuando se tienen que hacer consultas complejas en los datos.

Columnas

1. Acceso mediante clave como las Clave-Valor, pero agrupada por columnas.
2. Tabla, con filas y columnas, similar a las relacionales, pero muy distinto en implementación.
3. Los datos se almacenan por columnas.
4. No es costoso agregar columnas.
5. Cada fila puede tener un conjunto distinto de columnas.
6. Las tablas pueden tener valores nulos, sin costar espacio.

Ejemplos

1. HBase
2. Cassandra
3. Hypertable
4. Google BigTable.

Cuando es bueno

1. Escalabilidad horizontal.
2. Compresión y versionado (apropiado para indexar web pages).

Cuando no es tan bueno

1. Cuando no se sabe cómo se van a usar los datos. El diseño del esquema se beneficia si se sabe cómo se van a usar los datos.

Documentos

1. Esquemas flexibles.
2. Objetos anidados.
3. Facil de distribuir.
4. Los objetos no estan relacionados, estan anidados.
5. Desnormalización de datos.

Ejemplos

1. MongoDB
2. CouchDB
3. OrientDB

Cuando es bueno

1. Cuando no se sabe como van a ser los datos.
2. Mapean bien a modelos de programación orientados a objetos.

Cuando no es tan bueno

1. Cuando se van a querer hacer consultas complejas con joins. Cada documento deberia contener toda la información que generalmente se va a querer sobre él.

Grafos

1. Datos almacenados en forma de grafo.
2. Los datos se navegan mediante nodos y aristas.
3. Los nodos y las relaciones pueden tener propiedades (pares claves/valor).

Ejemplos

1. Neo4J
2. InfiniteGraph

Cuando es bueno

1. Aplicaciones donde hay redes.
2. Datos muy interconectados.

Cuando no esta tan bueno

1. Dificil de particionar una misma red. Si es muy grande la red, se puede usar el grafo para almacenar las relaciones y los datos se almacenan en otra db.

Base de Datos NoSQL

Riak

General

1. Base de datos de clave/valor distribuida.
2. Los valores pueden ser cualquier cosa (texto plano, JSON, XML, imágenes, videos, etc)
3. Interface HTTP REST (consultas via URLs, headers y verbos http, respuestas http standards)
4. Inspirado en Amazon Dynamo.

Features

1. Keys en buckets/namespaces para evitar colisiones de claves.
2. Links: metadatos que asocian una clave con otras, con una etiqueta.
3. Link walking: puedo traer los datos de las claves asociadas.
4. Permite agregar metadata usando headers con el prefijo X-Riak-Meta.
5. Soporta content-type de los datos.
6. Mapreduce con JavaScript o Erlang.
7. Permite almacenar funciones y trae alguna predefinidas.
8. Permite filtros de clave en los map.
9. Pre/Post commits hooks. Funciones JavaScript o Erlang.
10. Riak search: índice invertido que se puede activar.
11. Permite índices secundarios cambiando el storage_backend a riak_kv_eleveldb_backend.

Infraestructura

1. Todos los nodos son iguales.
2. Setear cantidad de particiones, cuantos nodos, cada particion maneja un rango de claves.
3. Tolerante a fallas, sin punto único de fallas.
4. N, W y R. N: cantidad de nodos que una escritura se replica. W: cantidad de nodos que deben ser escritos exitosamente para devolver una respuesta exitosa. R: es la cantidad de nodos que se necesitan leer un valor exitosamente. Modifica CAP.
5. Estos nros se pueden setear por request.
6. Tambien existe DW (durable write) (write en realidad guarda en memoria)
7. Si un server se cae, momentaneamente otro toma sus escrituras.
8. Vector clocks para resolver conflictos de escritura.

Fortalezas

1. Alta disponibilidad, sin único punto de fallas.

Debilidades

1. Consultas.
2. Solo Erlang para algunas funciones o mayor velocidad que JS.

Hbase

General

1. Base de datos por columnas.
2. Basado en BigTable.
3. Construido sobre Apache Hadoop (plataforma de software escalable y distribuido). En general se utiliza junto a algunos o varios de estos módulos.
4. Rápido para consultas en grandes bases de datos.
5. Tiene esquemas.
6. Tablas: mapas de mapas (clave/valor). Clave es un string que cada uno mapea a una fila. Cada fila es un mapa, donde cada clave es llamada columna y cada valor es un array de bytes.
7. Las columnas se agrupan en familias.
8. Util para casos de escala, no para casos simples (se dice que 5 nodos es lo mínimo que tiene sentido utilizar)

Features

1. Varios protocolos para conexión con los clientes.
2. Write-ahead logging.
3. Mapreduce con Java.
4. Versionado (timestamps cada vez que se guarda algo)
5. Compresión en campos.
6. Bloomfilters para búsquedas más rápidas de claves.
7. Consistencia
8. Hbase shell

Infraestructura

1. Tolerante a fallas.
2. Existe un master server.
3. Un conjunto de filas ordenadas forman una region. Cada region esta asignada a un servidor.
4. CP. Consistencia. Tolerante a particiones.

Fortalezas

1. Escala bien para grandes datos.
2. Comunidad.

Debilidades

1. No scale down.
2. Alto mantenimiento, muchos modulos involucrados, complejo.
3. No índice más alla que la clave de fila.

4. Sin tipos de datos.

MongoDB

General

1. Base de datos de documentos.
2. Sin esquemas
3. Colecciones

Features

1. JSON (BSON)
2. Linea de comandos
3. Motor de consultas con Javascript.
4. Se puede crear funciones personalizadas.
5. Objetos anidados.
6. Las consultas soportan los objetos anidados (any, all, etc)
7. Consultas booleanas y otras funciones predefinidas
8. Varios comandos para actualizar documentos. Actualizar solo un campo, incrementar un campo, agregar un valor a un array, etc.
9. Referencias (aunque no ayuda para hacer joins)
10. Creacion de índices
11. Consultas de agregación
12. Posibilidad de ejecutar funciones JS en el server.
13. Se pueden guardar funciones en el server.
14. Mapreduce.
15. GeoSpatial queries (index geohash)

Infraestructura

1. Replicacion y sharding
2. Un server es el master, pero si se cae, algun otro toma su rol (se vota, tambien puede haber un server arbitro). Los datos que no se replicaron se pierden.
3. Se puede ajustar CAP con parámetros de configuración.

Fortalezas

1. Soporta grandes cantidades de datos y gran cantidad de requests con replicación y escala horizontal.
2. Flexible modelo de datos.
3. Facil de usar viniendo de SQL.
4. Util para casi cualquier aplicación.

Debilidades

1. Cuidado con typo de nombres de campos y colecciones.
2. Diseño para evitar joins.
3. Duplicación de datos.

Neo4J

General

1. Base de datos de grafos.
2. Almacena los datos como un grafo.
3. Whiteboard friendly: si se puede dibujar un diseño como cajas y lineas en un pizarron, se puede almacenar en Neo4J.
4. Se enfoca más en las relaciones entre valores que en los cosas comunes entre los conjuntos de valores.
5. Puede almacenar grandes cantidades de nodos y aristas.
6. Sin esquemas

Features

1. Web interface
2. Lenguajes para navegar nodos y aristas: Java, REST, Cypher, Ruby, Gremlin.
3. Gremlin, sobre Groovy, lenguaje estilo pipes (vinos similares ice_wine.out('grape_type').in('grape_type').filter{ !it.equals(ice_wine) })
4. inE, outE, inV, outV, bothE, bothV
5. except, loop, dedup
6. groupCount
7. Mapreduce like queries con collect() e inject()
8. DSL en Gremlin (ej, alumno curso materia, out(“curso”).dedup -> curso)
9. Algoritmos para recorrer el grafo. Desde, hasta, shortestPath, allPaths, dijkstra.
10. Permite crear índices.
11. Full text inverted index.
12. Algoritmos para dar puntaje a los nodos y otros se pueden obtener en JUNG (Java Univeral Network/Graph) Framework.
13. ACID

Infraestructura

1. Soporte a clusters, con replicación master/slave.
1. Alta disponibilidad mediante slaves. Eventualmente consitente.
2. Neo4J HA cluster. Zookeeper de Apache Hadoop como cluster coordinador.
3. Si se cae el master, un slave pasa a ser master.
4. HA es AP. Eventualmente consistente.

Fortalezas

1. Sin esquemas.
2. Sin limitaciones en cómo los datos estan relacionados.
3. La navegación en el grafo es muy rápida y no depende de la cantidad de datos en el grafo.
4. Alta disponibilidad con HA.
5. El diseño de grafo es intuitivo.

Debilidades

1. No se puede shardear.
2. GPL para la versión Community, sin HA, la Enterprise es paga.

Links Relacionados

1. http://nosql-database.org/
2. http://pragprog.com/book/rwdata/seven-databases-in-seven-weeks