BIG DATA

STORING AND RETREIVING LARGE DATA VOLUMES

1. DATABASE

La base de dades més simple està formada per claus i valors (key, value). Si necessitem un valor, demanem la seva clau amb get(key).

- db_set(key, val): afegeix al final del fitxer (BD). Si sobreescric una clau, al fer el get em retornarà l'última.
- db_get(key): busca la paraula a la BD i es queda amb l'última.

Aquesta BD té un bon rendiment d'escriptura, perquè afegir al final del document és ràpid (log-file: seqüència de registres només per afegir). Però té un mal rendiment de lectura si la BD té molts registres, perquè cada consulta comença al primer element i llegeix cada element del fitxer des del principi fins al final (ja que es queda amb l'última instància de la clau que estem buscant). Per tant, té una complexitat de O(n), on n és el nombre de registres.

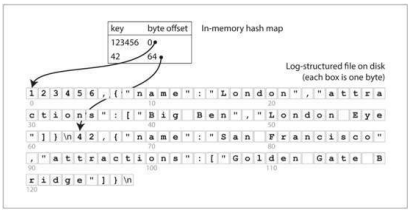
Exemples de complexitat: O(log n) - low cost, O(n*m) - high cost, O(n) - high cost, O(1) - low cost.

Per accelerar les lectures fem ús dels índex, la qual cosa ens permet cercar valors de manera eficient. Hem de generar i emmagatzemar metadades per trobar els valors ràpidament en una estructura de dades addicional derivada de les dades originals, de tal manera que no afecta les dades ni el contingut de la BD original. Però les actualitzacions dels índex afecten a les operacions d'escriptura de la BD, ja que cal afegir les dades i després actualitzar l'índex.

Tipus d'índex comuns: hash, sorted string tables, log-structures merge trees.

Hash

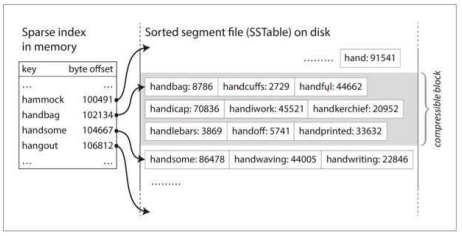
Mapa hash a la memòria que es guarda la posició de les claus en un fitxer (per guardar key-values). Cada element té dos valors, la clau i la posició. Al fer la lectura la capçalera del disc s'ha de col·locar a la ubicació de les dades (hora de cerca), i es llegeix la operació del valor del disc. Quan volem escriure en una mapa hash, afegim la clau i el valor al final del fitxer, i afegim una nova posició de clau al mapa.



Els principals problemes del mapa hash són que ha de cabre en memòria (com que el patró d'accés és aleatori pot haver col·lisions entre claus) i que no es pot aplicar en datasets amb moltes claus.

Sorted String Tables

Cada bloc té parells de clau-valor ordenats per clau. Cada bloc té només una instància de cada clau. Ara necessitem una nova operació: merge and sort (Si per exemple tenim 3 segments, agafem la clau més petita i la posem al fitxer de sortida i ho repetim). Per saber si existeix una clau no necessitem un índex de totes les claus a memòria, sinó que tindrem un índex per uns quants kilobytes, per tant, tindrem tantes claus com blocs.



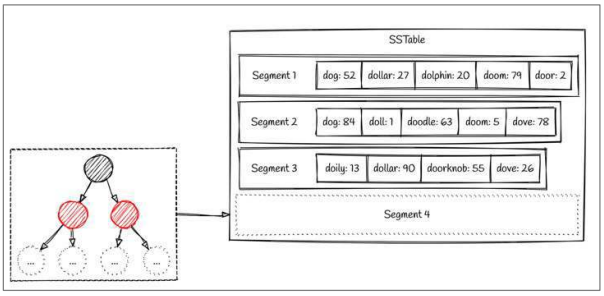
Per buscar una paraula, si la clau es troba a l'índex anirem a la posició del bloc, si no la trobem a l'índex, anirem al bloc anterior i buscarem la clau.

Log-structures merge trees (LSM Trees)

Per carregues de treball de moltes escriptures, on només s'accepten operacions d'escriptura seqüencials (amb això tindrem un bon rendiment d'escriptura). Estan darrere de moltes BD comercials (Google BigTable, AWS Cassandra, Scylla, RocksDB). Les dades estan al disc en SSTables, les quals estan formades per fitxers anomenats segments, que

són inmutables un cop s'han escrit al disc. A diferència del hash, tenim un bon rendiment per consultes d'interval cat0000-cat9999. Funciona bé quan el conjunt de dades és més gran a la memòria.

Quan s'escriuen nous valors (key, value), es guarden a la memtable (estructura de key-value ordenada a memòria). Quan aquesta està plena, s'escriu a disc com un nou segment de la SSTable. Per llegir les dades, començariem per llegir l'últim segment per trobar la clau, i si no la trobem, aniriem mirant els següents més recents, fins trobar-la. Això significaria que podríem retornar les claus que s'han afegit més recentment més ràpid. Una optimització senzilla és utilitzar el sparse índex a memòria (igual que a SST).



A mesura que passa el temps, aquest sistema anirà acumulant segments, els quals s'han de netejar i mantenir per tal d'evitar que hi hagi masses. Per aquesta raó compactarem les dades. Si tenim les mateixes claus en diferents segments, amb la compactació tindrem un únic segment amb la clau només una vegada.

Un problema que tenim és si el sistema falla, ja que totes les dades que estiguin a la memtable es perdran. Una solució seria tenim un altre log-file al disk sense un ordre aplicat, i quan el sistema falli, reconstruirem la memtable a partir d'aquest log-file afegint totes les claus i valors.

Limitacions:

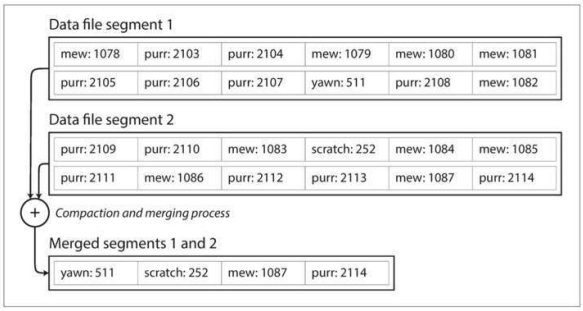
- La operació de compactació afecta al rendiment de lectura i escriptura, ja que les lectures s'han d'esperar a que acabi la compactació.
- S'ha de compartir la capacitat d'escriptura del disc amb tres operacions (logging, copiar la memtable a un segment, background compact).
- Write burst to memtable: la compactació no fa front a block merge requests, les lectures són més lentes perquè tenim més blocs sense compactar, i ens podem quedar sense espai al disc.

Espai al disc

Si ens quedem sense espai al disc, tenim tres opcions. La primera és trencar el fitxer de log en fitxers de mida fixada, que la següent operació s'escrigui a un altre fitxer, i finalment compactar els fitxers per estalviar espai.

Si tinc un fitxer amb claus i valors amb claus repetides, puc compactar les dades eliminant duplicats i guardant només l'últim clau-valor. En canvi, si tenim dos fitxers (dos blocs), podem generar nous blocs per substituir els antics, fent el mateix que hem dit abans (guardar només l'últim clau-valor entre els dos fitxers).

Cada segment té el seu mapa hash a memòria. Quan volem fer una cerca, busquem la clau al mapa hash més recent, si no està, al segon més recent, etc , fins trobar el valor. Hauriem de tenir un nombre limitat de segments per evitar operacions addicionals de lectura hash.



Té avantatges utilitzar un fitxer que només es pot afegir al final (immutable). Afegir i combinar són operacions seqüencials d'escriptura (és més ràpid que les operacions d'escriptura aleatòries, en qualsevol posició), a recuperació d'errors és senzilla d'implementar i les operacions de fusió de blocs gestionen els problemes de fragmentació a temps.

Casos d'arquitectura key-value

- KingDB: log-structured storage
- LevelDB: LSM-Trees

BIG DATA

DATA WAREHOUSING

1. OLTP i OLAP

Inicialment les operacions eren transaccions comercials, com per exemple fer una venda, fer una comanda o pagar el sou a un empleat. Però les aplicacions es tornen més complexes i la definició de transacció passa a ser més genèrica, com per exemple un comentari en un blog, una acció en un joc o el funcionament local de NFC.

OLTP (OnLine Transaction Processing) és un sistema de tractament de dades que s'utilitza per gestionar un gran nombre de transaccions curtes en línia. Permet l'execució en temps real d'un gran nombre de transaccions de bases de dades per part d'un gran nombre de persones, normalment a través d'Internet. Admet un processament molt ràpid, amb temps de resposta mesurat en ms. Proporciona conjunt de dades indexats per a una cerca ràpida.

Amb OLTP, les aplicacions cerquen un nombre reduït de registres, i aquests s'insereixen o s'actualitzen. Les dades es carreguen cada cop que es fa una transacció.

Propietats d'una base de dades per garantir la fiabilitat de les transaccions (**Transactional workloads**):

- **Atomicitat**: cada transacció es tracta com una única unitat, la qual triomfa o fracassa completament.
- **Consistència**: les transaccions només poder agafar dades de la base de dades d'un estat vàlid a una altre.
- **Aïllament**: l'execució simultània de transaccions deixa la BD en el mateix estat.
- **Durabilitat**: si es confirma una transacció, el resultat d'aquesta és definitiu.

Exemples: cançons preferides a Spotify, notes acadèmiques, cerca de gens NCBI per nom de gen, últimes 10 operacions del meu compte bancàri.

Si volem obtenir anàlisis de les dades, hem d'utilitzar **OLAP** (OnLine Analytic Processing). Escaneja un gran nombre de registres, llegeix poques columnes per registre, calcula estadístiques d'agregació (ja que les consultes sovint son molt complexes i necessiten agregacions) i no retorna dades no processades a l'usuari. Serveix per fer anàlisis de dades amb grans volums de dades. Normalment aquestes dades provenen d'un data warehouse. Les dades es carreguen periòdicament, s'agreguen i es guarden en un cub.

Analytical workloads: s'utilitzen per a l'anàlisi de dades i la presa de decisions.

- Resums
- Tendències
- Informació empresarial

Exemples: ingressos totals d'un mes, quants productes més es van vendre el passat Black Friday, quantes persones han mort per covid des de la pandèmia.

Propietat	OLTP	OLAP
Patró de lectura	Pocs registres per consulta obtinguts per clau	Agregació d'un gran nombre de registres
Patró d'escriptura	Escriptures d'accés aleatori i de baixa latència	Bulk import (ELT) o flux d'esdeveniments
Utilitzar per	Pel client, mitjançant l'applicació web	Anàlisi intern, per prendre decisions
Dades	Últim estat de les dades	Historial d'events que van passant al llarg del temps
Mida del dataset	Gigabytes to Terabytes	Terabytes to Petabytes
Tipus de dataset	Bases de dades tradicionals (DBMS)	Data warehouse

Ona Folch

El processament de dades (**Data Processing**) és la conversió de dades sense processar a informació significativa mitjançant un procés. Hi ha dos tipus:

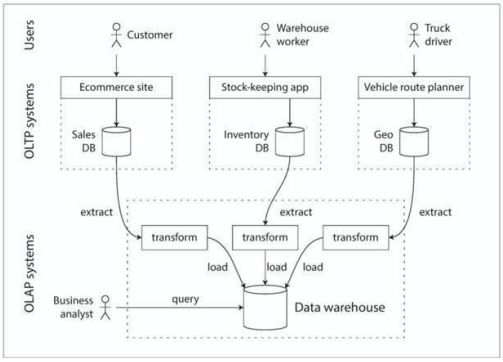
- Batch processing: les dades es recullen en grups, i en un futur es processa tot el grup com un batch.
- Stream processing: cada nova dada es processa quan arriba.

Data Warehouse

S'utilitza menys ús dels sistemes OLTP per l'anàlisi de dades, i com a alternativa s'executen les anàlisis en una base de dades independent (Data Warehouse). No afecta la BD principal, ja que 'aboca' totes les dades a una nova BD per analitzar-les, la qual està preparada per rebre preguntes intensives de dades. Concentra i emmagatzema de forma estructurada la informació obtinguda per poder analitzar-la fàcilment. Són dades que, encara que no estan disponibles a temps real, poden ser analitzades de forma ràpida i massiva sense interrompre els processos de l'usuari.

Construir Datsets per OLAP (ETL, Extract, Transform, Load)

Com crear grans conjunts de dades per l'anàlisi. Operacions principals per aplicar als datasets: extreure les dades de la BD de OLPT, transformar les dades en un esquema per facilitar l'anàlisi, netejar i filtrar errors i carregar les dades al warehouse.



Separem les dades del warehouse amb les de OLTP per optimitzar el procés d'anàlisi, i perquè els algoritmes d'indexeció de OLTP no són adequats per consultes analítiques. Proveïdors de data warehouse: Teradata, Vertica, SAP HANA, ParAccel, AWS Redshift.

Amazon Redshift: Sistema de gestió i consulta de BD relacionals i data warehouse de classe empresarial. Operacions de consulta que recuperen, comparen i avaluen grans quantitats de dades. L'emmagatzematge i el rendiment està optimitzat (processament massiu paral·lel).

The data jouney: Primer de tot s'han d'obtenir les dades i importar-les a una base de dades (**Data Ingestion**), un cop les tenim a la BD les processem i les convertim en un format més significatiu amb ETL o ELT (**Data Processing**), i finalment creem representacions gràfiques de la informació (**Data Visualization**).

Serveis d'azure pel data warehouse:

- Azure Data Factory: és un servei per la integració i transformació de les dades. Recupera dades de més d'una font d i les filtra per treure el soroll i quedar-se amb la informació important. Es va executant a mesura que es reben les dades.
- Azure Data Lake: és un repositori de dades, el qual organitza les dades en directoris per millorar l'accés als fitxers.
- Azure Databricks: és una plataforma basada en apache spark que proporciona processament i transmissió de big data.
- Azure HDInsight: és un servei de processament de big data que ens permet utilitzar biblioteques de codi obert en una plataforma, en un entorn Azure.

Ona Folch

2. DATA SYSTEM ROLES

Administrador de la BD: gestiona la BD, implementa còpies de seguretat de les dades, controla l'accés dels usuaris i supervisa el rendiment.
Eines comunes que utilitza:

- Azure Data Studio: Interfície gràfica per gestionar serveis de dades locals i basats en núvol (Windows, macOS, Linux).
- SQL Server Management Studio: Interfície gràfica per gestionar serveis de dades locals i basats en núvol (Windows).
- Azure Portal/CLI: Eines per a la gestió i el subministrament d'Azure Data Services.

Enginyer de dades: processa les dades, prepara les dades per ser analitzades i du a terme l'emmagatzenament d'ingestió de dades (procés de transport de dades d'una o més fonts a un lloc objectiu per a un posterior processament i anàlisi).
Eines comunes que utilitza:

- Azure Synapse Studio: Azure Portal integrat per gestionar Azure Synapse, ingestió de dades.
- SQL Server Management Studio: Interfície gràfica per gestionar serveis de dades locals i basats en núvol (Windows).
- Azure Portal/CLI: Eines de gestió i provisió de recursos.

Analista de dades: proporciona informació sobre les dades de manera visual, modela les dades per poder-les analitzar i combina dades per visualitzar-les i analitzar-les.
Eines comunes que utilitza: (Power BI és un servei d'analítica empresarial de Microsoft)

- Power BI Desktop: eina per visualitzar dades.
- Power BI Portal/Power BI Service: per elaborar i gestionar informes de Power BI. Per compartir datasets i informes.
- Power BI Report Builder: eina per visualitzar dades, i visualitzar i modelar informes.

BIG DATA

REDIS

1. REDIS

Les dades estan emmagatzemades en un únic espai d'adreces, que és la memòria, i per tant es redueix la complexitat de la solució i no cal paginar informació for de la memòria, ni dissenyar mètodes de coherència. Tenim un accés ràpid i les estructures de dades ja no s'han d'optimitzar per al disc.

Principals problemes a resoldre:

- **Durabilitat**: què fer quan el sistema s'apaga? Hauríem de crear i distribuir còpies de dades i incrementar la redundància.
- **Capacitat**: què fer quan les necessitats de dades són més grans que l'espai de memòria? Hauríem d'afegir més servidors al sistema, de tal manera que tindríem els blocs de dades distribuïts entre diferents servidors, no són còpies.

Al afegir nous servidors, tenim nous problemes. Quan fem consultes complexes, haurem de llegir totes les dades distribuïdes entre els servidors, i necessitarem una distributed global join (si per exemple tenir dos servidors i volem trobar el màxim d'algun valor, haurem de buscar en els dos i fer un join per saber quin és el màxim entre els dos elements).

L'ús de dades locals a les bases de dades NoSQL eviten la necessitat de fer consultes join entre particions.

Els avenços tecnològics van millorant, però la freqüència i la capacitat d'un microprocessador CPU no ha avançat, però ens permeten tenir sistemes de bases de dades amb processadors de molts nuclis.

Aquests avenços ens permeten guardar les dades a memòria si el nostre conjunt de dades hi cap, ja que és més ràpid que al disc. La memòria + SDD s'ha convertit en un nou estàndard (ja està predeterminat els serveis de BD al núvol), i la majoria de BD no són tant grans.

Si el conjunt de dades no cap a memòria, no podem utilitzar redis, ja que s'utilitza la memòria com a emmagatzematge principal.

El disc només s'utilitza per fer persistència (per fer còpies, ja que si el sistema cau, es perdran poques dades) i està orientat a l'estructura de dades (és més complex que el sistemes de clau-valor). Redis és open source, i ens permet donar informació als usuaris en temps real (amb poca latència).

No s'utilitzen taules, es guarden objectes individuals a memòria.

Ona Folch