Výhody výpočtov v prostredí in-memory databáz

Neustály vývoj a zlacňovanie počítačového hardvéru má za následok, že technológia, ktorá bola pred pár rokmi nemysliteľne drahá, dnes sa stáva dostupnou.

Do tejto kategórie spadá aj počítačová pamäť. Kým pred pár rokmi mať všetky dáta v pamäti bolo nemožné, dnes už existujú počítače s 1,5 TB pamäťou. (1) Treba sa teda zamyslieť, či spôsob ukladania a spracovania dát, ktorý bol vymyslený pred viac ako tridsiatimi rokmi, je naozaj najlepší pre dnešné množstvo dát a či je vhodný na ich spracovanie a analýzu v reálnom čase.

Práve nutnosť analyzovať dáta v reálnom čase je jednou z najväčších tém súčasnosti. Môže za to najmä rozmach internetu a sociálnych sietí. Marketéri sa snažia čo najlepšie predpovedať správanie používateľov a podľa toho určiť, akú reklamu komu zobraziť. Práve v tejto situácii musia bojovať s časom, keďže majú len niekoľko stotín sekúnd. Počas nich sa musia rozhodnúť, ktorú z reklám zobrazia, kým sa načíta stránka.

Podobne je to aj s údajmi zo sieťovej prevádzky. Ak chceme zabrániť počítačovému útočníkovi v tom, aby napáchal viac škôd, musíme zareagovať čím rýchlejšie. V opačnom prípade môže zaútočiť na iné počítače či ukradnúť viac údajov.

Musíme teda zrýchliť analýzu dát a prístup k nim. Aby sme videli, prečo je nutné ich držať v pamäti, pozrime si nasledujúcu tabuľku. Tá popisuje rýchlosť čítania z rôznych typov pamäťových médií. Ako môžeme vidieť, čítanie z pamäte je až 100 000-krát rýchlejšie ako z disku a 1 500-krát rýchlejšie ako z SSD.

L1 cache reference (cached data word)	**0.5ns**	
Branch mispredict	5ns	
L2 cache reference	7ns	
Mutex lock/unlock	25ns	
Main memory reference	**100ns**	**0.1µs**
Send 2K bytes over 1 Gb/s network	20,000ns	20µs
SSD random read	150,000ns	150µs
Read 1 MB sequentially from memory	250,000ns	250µs
Disk seek	**10,000,000ns**	**10ms**
Send packet CA to Netherlands to CA	150,000,000ns	150ms

Z tabuľky vidíme, že ak dáta máme v pamäti, dosiahneme obrovské zrýchlenie. Má to však aj svoje nevýhody. Napríklad operačná pamäť je volatilná, teda všetky dáta sa stratia po vypnutí počítača a zvýšia sa nám náklady na hardvér. Pozrime sa na najpoužívanejšie in-memory databázy, na ich výhody a nevýhody a spôsob, akým riešia spomínané problémy.

Memcached

Memcached je key-value dátová štruktúra uložená v pamäti. Využíva sa ako cache na zrýchlenie databázových dopytov, najmä dynamickými webovými aplikáciami. Teda nejde priamo o databázu v pravom slova zmysle, keďže samotné Memcached nerieši perzistenciu dát.

Key-value znamená, že každý jeden „riadok“ je uložený vo forme jedinečného kľúča a dát, ktoré môžu byť takmer ľubovoľného formátu. Dáta sú zoradené podľa kľúča, čo veľmi zrýchľuje prístup k nim. Memcached pridáva jeden atribút navyše, konkrétne čas exspirácie, ktorý je nutný najmä kvôli svojmu primárnemu využitiu ako cache.

Memcached znižuje zaťaženie databázy tak, že si ukladá výsledky dopytov do svojej štruktúry nasledujúcim spôsobom:

1) Ak klient požiada o nejaké dáta (napr. výsledok databázového dopytu alebo časť webstránky), Memcached sa pozrie, či už nemá uložené tieto dáta.
2) a) Ak tieto dáta má, tak nie je potrebné zavolať databázový dopyt, dáta sa vrátia z Memcached.
   b) Ak tieto dáta nemá uložené, tak sa zavolá databázový dopyt, ale výsledok sa uloží aj v Memcached.
3) Ak sa niektoré dáta v databáze zmenia alebo exspiruje ich platnosť, Memcached musí updateovať svoj cache. (2)

Redis

Redis je in-memory dátová štruktúra, ktorá sa používa ako databáza, message broker alebo cache. Spadá do kategórie NoSQL databáz. Primárne využíva key-value model, ale podporuje aj dokumentový, grafový model a spracovanie časových údajov. Práve preto je často využívaná na zrýchlenie aplikácií, napr. na načítavanie webstránok. Jej popularita spočíva najmä v jej všestrannosti:

1) Nie je to len cache, ale dá sa využívať aj ako perzistentná databáza vďaka logovaniu na disk. Napriek tomu, že na prvý pohľad hneď strácame rýchlosť in-memory výpočtov, databáza sa nám nespomalí v takej miere, ako by sme predpokladali. Je to spôsobené tým, že jednak všetky read operácie sa udejú len v pamäti, jednak všetky write operácie sa zapisujú len sekvenčne do logov, kde sú disky kvôli sekvenčnému zápisu veľmi rýchle (viac ako 300 MB/s). Je možné zvoliť si viacero logovacích techník:
   a) RDB: je to jeden súbor, reprezentujúci celú databázu v danom časovom okamihu. Vytvára sa v každom používateľom zadanom časovom intervale. Využíva sa najmä na backup databázy. (3)
   b) AOF: systém zapíše každú jednu write operáciu na disk, ktoré sa po jednom spustia pri backupe servera. Takto sa postupne rekonštruuje databáza. Je to oveľa bezpečnejší spôsob logovania dát ako RDB, keďže všetko sa hneď uloží na disk, ale logovacie súbory môžu rýchlo dosiahnuť veľmi veľkú veľkosť.
2) Databáza podporuje veľa dátových typov, ako napr. listy, polia, množiny a usporiadané množiny.
3) Hodnoty v key-value štruktúre môžu mať až 512 MB, čo je v porovnaní s 1 MB pri Memcached významným plusom pre túto databázu.

SAP HANA

SAP HANA je nová in-memory relačná databáza od firmy SAP, ktorá sa primárne zaoberá vývojom ERP systémov pre veľké podniky. SAP HANA je taktiež jediná databáza, ktorá je podporovaná pre ich nový ERP produkt S/4HANA.

Nie je to len samotná databáza, ale aj platforma, ktorá má rôzne rozšírenia, ako je vlastný aplikačný server, podpora pre streamované dáta, strojové učenie, analýza dát v reálnom čase a ďalšie. Práve nutnosť dátových analýz v reálnom čase viedla k vzniku tejto databázy.

Pracovníkom SAP-u sa nepáčilo, že firmy majú svoje dáta v dvoch rôznych databázach, v jednej procesné dáta a v druhej dáta pre analýzu.

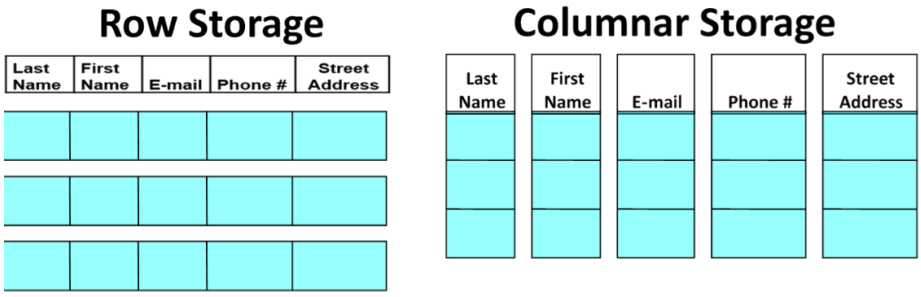
Aplikácie, ktoré pracujú s dátami, môžeme rozdeliť do dvoch veľkých skupín:

a) OLTP (On-line Transaction Processing): tieto aplikácie sú charakterizované veľkým počtom krátkych transakcií, ako write operácií (update, insert a delete). Tieto aplikácie slúžia na zabezpečenie chodu firmy, ako napr. na vytváranie objednávok, správu účtovníctva, kontrolu skladov a pod. S týmito systémami väčšinou pracuje veľa používateľov. Tí zadávajú veľké množstvo dopytov, ktoré ale nie sú výpočtovo zložité. Efektivita databáz je určená najmä počtom transakcií za sekundu. Dáta sú väčšinou uložené v 3. normálnej forme.
b) OLAP (On-line Analytical Processing): tieto aplikácie slúžia na analytické účely. Z toho vyplýva, že používateľov týchto aplikácií je menej a zadávajú aj menej dopytov. Tie sú ale zvyčajne komplexné, vyžadujú agregácie, trvajú dlho a skoro všetky dopyty si vyžadujú len read operácie. Asi najlepší príklad na tento typ aplikácií sú dátové sklady. Typicky dáta nie sú normalizované, využívajú sa multidimenzionálne, star alebo snowflake schémy (hviezda, vločka). Hlavnou myšlienkou týchto schém je, že atribúty rozdelíme do dvoch skupín. V jednej sú tie, ktoré sa dajú merať, ako napr. počet predaných kusov, cena. V druhej sú tie, ktoré len popisujú niečo, ako napr. farba, miesto predaja a pod. Na základe tohto rozdelenia sa potom vytvorí tabuľka faktov a dimenzie, ktoré sú spojené cudzími kľúčmi. Dáta v dimenziách sú väčšinou duplicitné, ale keďže je ich malý počet, nerobí to veľký problém. Keďže dáta sú zväčša určené len na čítanie, nenastane ani problém s dátovou integritou.

Rozdelenie týchto aplikácií (a aj databáz, s ktorými pracujú) má ale jednu veľkú nevýhodu. Dáta sa dostávajú do dátového skladu len raz za čas, teda sa analyzujú vždy len staré dáta. Ďalej, dáta treba predspracovať, aby mohli poputovať do dátového skladu. Tento ETL proces je zväčša zdĺhavý, vyžaduje si veľa ľudskej práce a dáta máme „zbytočne“ v dvoch rôznych kópiách. Analýza v OLTP systémoch je zvyčajne pomalá, čo nám opäť znemožní robiť analýzy v reálnom čase. Tento problém rieši práve SAP HANA. Uveďme si hlavné princípy jej fungovania:

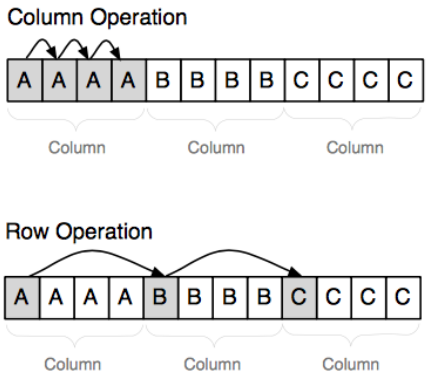
Stĺpcové ukladanie dát (4)

Stĺpcové ukladanie dát (columnar data layout) je jedným z hlavných konceptov tejto databázy. V tradičných databázach sú dáta uložené v riadkovom formáte (row data layout), teda dáta z jedného riadku, kým pri stĺpcovom formáte dáta z jedného stĺpca sú uložené za sebou v pamäti.

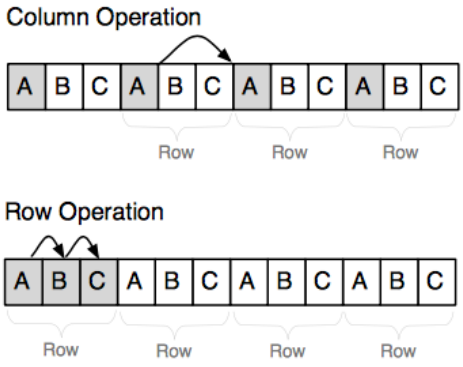


Teda ak chceme prečítať dáta z jedného stĺpca, tak pri stĺpcovom formáte jednoducho prečítame dáta za sebou, pri riadkovom formáte musíme „preskakovať“ ďalšie hodnoty z daného riadka. Ak ale chceme prečítať všetky dáta z jedného riadku, pri stĺpcovom formáte musíme „skákať“, kým pri riadkovom formáte sekvenčne prečítame jednu časť pamäte.

Columnar data layout



Row data layout



Takéto ukladanie dát má mnoho výhod najmä z analytického hľadiska. Zrýchlia sa agregácie v rámci jedného stĺpca a zrýchli sa aj filtrovanie. Aj keď sa na prvý pohľad javí, že čítanie je rovnako rýchle, lebo sme v operačnej pamäti a nie na disku, teda „skoky“ v pamäti nám nerobia problém, nastane tu iný problém, tzv. cache-misses. Aby procesor vedel pracovať s dátami, tie musia byť v L1 cache-i. Aby sa tam dali načítať, najprv musia byť načítané do L2, L3, L4 cache-ov. Dáta z operačnej pamäte môžu byť načítané iba v 64 byte-ových blokoch. To má za následok, že zbytočne načítame nepotrebné dáta do cache-ov. Ukážme si to na jednoduchom príklade. Chceme sčítať všetky hodnoty v jednom stĺpci, nech sú to celé čísla s veľkosťou 4 byte-y. Teda na načítanie každej hodnoty potrebujeme načítať 16-krát toľko dát, koľko reálne potrebujeme, teda nám to 16-násobne spomalí proces.

Stĺpcový formát dát sa aj v minulosti využíval na analytické účely, ale kvôli pomalým rekonštrukciám riadkov a pomalým insert/update operáciám bol nepoužiteľný pre OLTP aplikácie. Tento nedostatok sa čiastočne eliminuje tým, že kým na disku tento proces spomaľuje najmä pomalé posúvanie čítacej hlavy, v operačnej pamäti čítanie z každej časti pamäte je rovnako rýchle.

Výhody stĺpcového formátu narastajú spolu s počtom dát. Môžeme mať milióny riadkov, na ktorých robíme analýzy, naraz ale zobrazíme maximálne pár stoviek riadkov

jednej tabuľky. Viac už človek nedokáže prečítať ani spracovať, teda tento počet zostáva aj napriek rastúcemu počtu dát takmer konštantný.

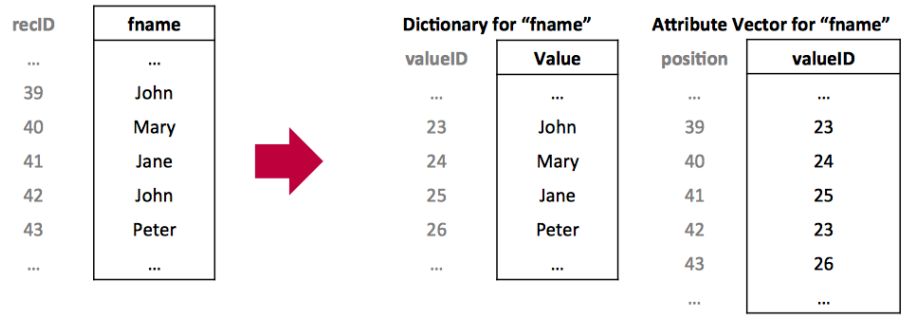
Dictionary encoding (4, 5)

(ukážkový príklad sa môže urobit pre tabulku sveta, Slovenska, pre nase data- tu je jediny problem, ze potom nase data je nutme skor uz uviest do kontextu)

Stĺpcový formát dát nám umožňuje zaviesť nové kódovanie. Čo je dôležité, táto kompresia, na rozdiel od tradičných metód, nám umožňuje pracovať priamo s komprimovanými dátami. Pri iných metódach je nutné najprv dáta dekódovať, aby sme s nimi vedeli pracovať. Takto dostávame 3 hlavné výhody tejto metódy:

a) Potrebujeme menej pamäte na uskladnenie dát, čo znižuje náklady na drahú operačnú pamäť.
b) Nepotrebujeme dekódovať dáta pred spracovaním, čím ušetríme čas, teda celý proces sa zrýchli.
c) Je potrebné načítať menej dát z pamäte do cache-ov, čo je jedna z najpomalších častí celého výpočtu, teda výpočet sa zrýchli ešte viac.

Kódovanie je založené najmä na malej diverzite dát v jednom stĺpci. Napr. zoberme si tabuľku obyvateľov Slovenska. Kým v tabuľke je 5 miliónov riadkov, krstných mien je pár stoviek. Každý stĺpec nahradíme „slovníkom“ a stĺpcovým vektorom, teda každému krstnému menu priradíme jedno číslo a v pôvodnej tabuľke namiesto mena uvedieme len toto číslo.



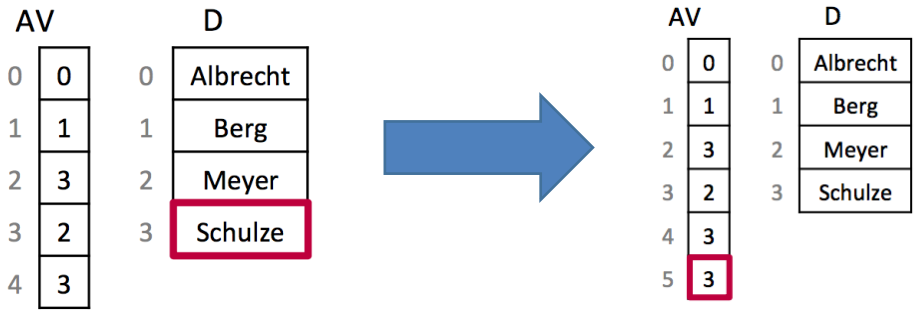
Nech jedno krstné meno pozostáva z 8 znakov, teda na jeho zakódovanie potrebujeme 16 bytov. Teda v pôvodnej tabuľke krstné mená zaberali $5\,000\,000 * 16 = 80\,000\,000$ bytov. Keďže rôznych mien je menej ako 1024, na zakódovanie jedného nám stačí 10 bitov, čo je 1,25 bytov. Atribútový vektor teda bude zaberať $5\,000\,000 * 10 = 50\,000\,000$ bitov, čo je 6 250 000 bytov. Pre slovník potrebujeme ďalších $1\,000 * (16 + 1{,}25) = 17\,250$ bytov, teda spolu je to 6 267 250 bytov. Dosiahli sme teda 12-násobné zníženie veľkosti.

Ďalšou výhodou je aj to, že sa zrýchlia dopyty s where podmienkami aj spájanie tabuliek (join) cez nečíselné atribúty. Dôvod je ten, že v tabuľkách sú všetky hodnoty v atribútových vektoroch celé čísla, s ktorými počítač vie pracovať najrýchlejšie. Napríklad, porovnanie dvoch reťazcov trvá násobne rýchlejšie ako dvoch celých čísel. Ďalšie zrýchlenie získame, ak sú dáta v slovníku zoradené (čo je aj defaultné nastavenie v SAP HANA). Každý stĺpec je skoro index, keďže kód danej hodnoty vieme vyhľadať v čase O(log n).

Nevýhody tohto prístupu sa prejavujú pri write operáciách a pri rekonštrukcii riadkov. Kým v tradičných databázach je jednoduché prečítať jeden riadok, v HANE musíme pospájať dáta. Prvý problém je, že dáta z jedného riadka máme v pamäti uložené na rôznych miestach. Druhý je ten, že namiesto hodnôt máme v tabuľke, teda v atribútovom vektore, len čísla kódujúce tieto hodnoty. Aby sme dostali hodnoty, musíme sa pozrieť do slovníka, aká hodnota prislúcha danému kódu. Preto by sa v dopytoch mali používať len tie stĺpce, ktoré sú reálne ďalej použité (a nie „SELECT *“). Toto platí aj pre tradičné databázové systémy, tam ale tento prístup má oveľa menší negatívny dopad na výkon databázy.

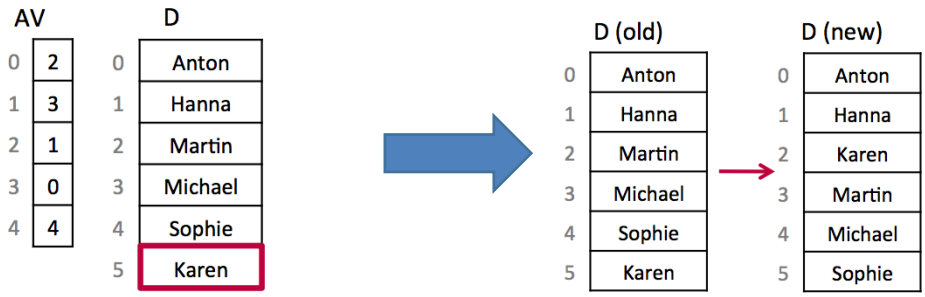
Problémy nám robia aj insert a update operácie. Ak chceme vložiť nový riadok, tak musíme vložiť novú hodnotu do každého stĺpca. Ak už daná hodnota je v slovníku, tak nám stačí vložiť novú hodnotu na koniec atribútového vektora.

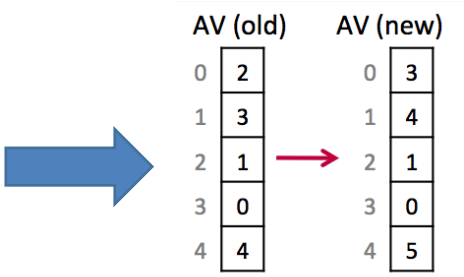
Napr. do tabuľky chceme vložiť človeka s priezviskom Schulze. Keďže slovník už obsahuje priezvisko Schulze, tak nám stačí vložiť jeho kód (3) na koniec atribútového vektoru.



Ak ale danú hodnotu ešte nemáme v slovníku, tak musíme vložiť novú hodnotu do slovníka. Keďže slovník je usporiadaný, tak ho musíme preusporiadať, čo ale má za následok, že sa zmenia kódy jednotlivých hodnôt, teda treba nanovo prepísať celý atribútový vektor.

Napr. do tabuľky chceme vložiť nového človeka s menom Karen. Keďže v slovníku nie je meno Karen, treba ho tam vložiť. Keďže slovník má byť usporiadaný, treba ho aj nanovo usporiadať.





Insert-only

Vidíme, že tento spôsob vkladania riadkov nie je efektívny. Pri vysokom počte write operácií by sa systém veľmi spomalil a stratili by sme všetky výhody, ktoré nám stĺpcový formát dát umožňuje. Podobný problém nastáva aj pri insert a delete operáciách. Môže sa stať, že musíme zmeniť slovník a nanovo usporiadať dáta.

Preto sa využíva iná technika. Delete operácie sú vyriešené tak, že ku každej tabuľke sa pridajú dva stĺpce: čas, odkedy je platný daný riadok a čas, dokedy je platný nový riadok. To znamená, že namiesto vymazania riadku sa len zmení čas, dokedy je platný daný riadok, z null hodnoty na časovú pečiatku daného okamihu.

Podobne ako v relačnej algebre, update je vymenený na kombináciu insert + delete. Keďže aj delete vlastne len zapisuje do databázy, tak databáza je vlastne len v insert-only móde. Tieto dva stĺpce majú ďalšiu výhodu, že dopyty „vidia“ len dáta, ktoré boli validné v danom okamihu, keď sa začal dopyt a nemusia sa robiť uzamknutia tabuliek. To je jedna z vecí, ktorá spomaľuje databázu pri veľkom počte používateľov pracujúcich s tou istou tabuľkou.

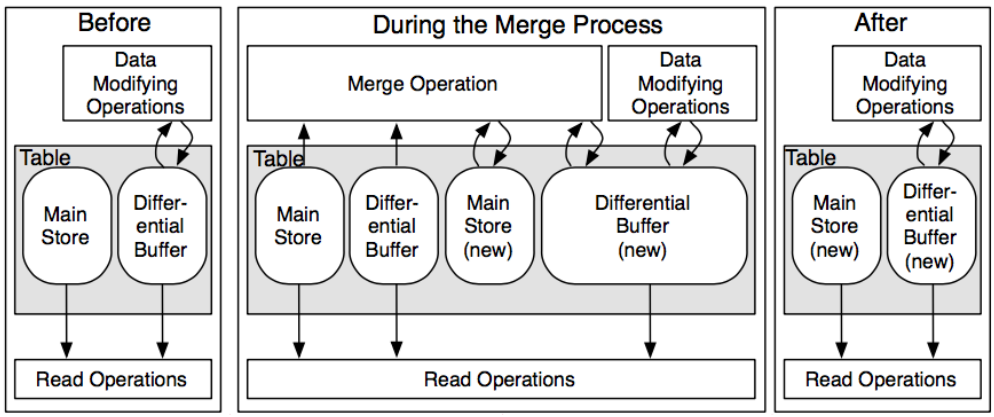
Differential buffer (delta storage)

Aby sa vyriešil problém s vkladaním nových údajov do tabuliek, zavedie sa rozdielový buffer (differential buffer). Je to malá tabuľka (do 50 000 riadkov), ktorá je tiež v stĺpcovom formáte, tiež sa využíva dictionary encoding, ale hodnoty v slovníku nie sú zoradené. Všetky nové dáta sa vkladajú do tohto buffera. Keďže nemusíme nanovo zoradiť slovník, inserty netrvajú dlho.

Všetky dopyty musia namiesto jednej tabuľky pracovať s hlavnou tabuľkou a rozdielovým bufferom. Keďže dáta v rozdielovom bufferi nie sú zoradené, tak nám to spomaľuje dopyty, aj keď nie o veľa. Avšak v bufferi je veľmi málo dát v porovnaní s hlavnou tabuľkou, preto toto spomalenie nie je signifikantné a agregačné dopyty sú aj takto oveľa rýchlejšie, ako keby sme dáta mali v riadkovom formáte. Vidno, že tento buffer funguje dobre, kým je relatívne malý. Preto ho treba z času na čas spojiť s hlavnou tabuľkou a urobiť takzvaný merge.

Merge

Databáza musí spojiť hlavnú tabuľku s delta úložiskom tak, aby systém bežal ďalej aj počas tohto spájania. Preto sa vytvorí nový prázdny rozdielový buffer a všetky nové zmeny sa budú zapisovať sem. Hlavná tabuľka a starý rozdielový buffer sa spoja do novej hlavnej tabuľky, ktorá vstúpi do platnosti, keď sa skončí spájanie. Potom sa môže vymazať stará hlavná tabuľka a starý rozdielový buffer. Najväčšou nevýhodou tohto prístupu je, že počas spájania potrebujeme dvakrát toľko pamäte, koľko zaberá samotná tabuľka.



Oracle Database In-Memory

Ako odpoveď na SAP HANA, najväčšia databázová firma s novou verziou svojej databázy Oracle Database 12c priniesla novú funkcionalitu Oracle Database In-Memory. (6) Hlavnou novinkou je uvedenie In-Memory Column Store (IM Column Store), čo znamená uloženie tabuľky v pamäti v stĺpcovom formáte, ktoré funguje na veľmi podobných princípoch ako stĺpcový formát dát v SAP HANA.

Hlavným rozdielom je, že dáta na disku sú vždy uložené v riadkovom formáte a všetky OLTP operácie pracujú s týmto riadkovým formátom, tak ako v tradičnej „diskovej“ databáze. Vytvorí sa druhá kópia dát, ktorá je celá v pamäti v stĺpcovom formáte. S touto kópiou pracujú analytické dopyty. Nevýhodou je, že máme dve kópie dát, teda nároky na pamäť sa výrazne zvýšia.

Výhodou je to, že my sa rozhodujeme, ktoré tabuľky budú uložené v oboch formátoch a vieme využiť výhody oboch formátov. Aj keď aj v tomto prípade pri zmenách v tabuľkách je nutné nanovo aktualizovať dáta aj v stĺpcovom formáte, teda aj túto databázu „spomaľuje“ stĺpcový formát pri write operáciách.

Prúdové spracovanie dát (streaming)

Každým dňom obrovským tempom rastie počet dát, ktoré ľudstvo produkuje. Toto tempo je také obrovské, že aj pri postupnom zlacňovaní hardvéru bude čoskoro viac pozbieraných dát, než dokážeme uložiť na všetkých počítačoch sveta. Veľa dát je vyprodukovaných priebežne a stále (bez prestávky) sa produkujú nové a nové dáta.

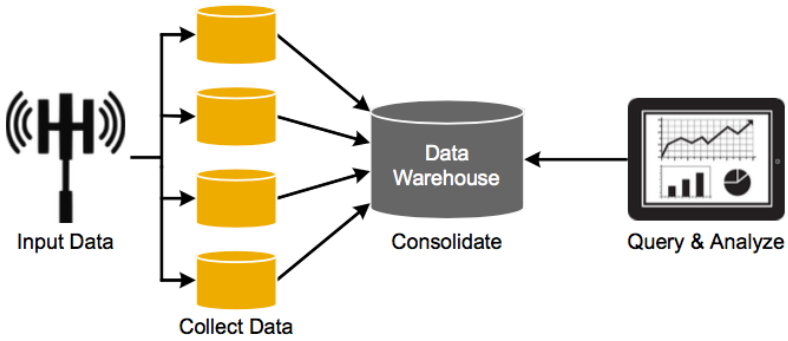
Tieto dáta (napr. z IOT senzorov) prichádzajú takým tempom, že musia byť hneď spracované, inak nebudeme stíhať spracovať ďalšie prichádzajúce dáta. Ďalším dôvodom je aj to, že veľakrát informácie z týchto dát chceme vedieť vyhodnotiť hneď, lebo staré informácie nám už nepomôžu. Napríklad, zhorí nám súčiastka, lebo sa prehriala a my sme na to prišli neskoro, alebo nám počítačový útočník ukradne všetky dáta, lebo sme ho neskoro odhalili.

Preto sa musí použiť nový typ algoritmov, spracovania dát, tzv. prúdové spracovanie, streaming. Dáta sa vyhodnotia a vyhodnotia hneď pri príchode, čo umožní ich analýzu v reálnom čase. Takto máme možnosť reagovať na dané udalosti hneď ako nastanú.

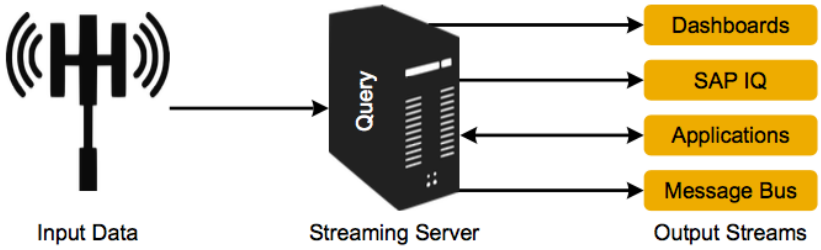
Umožní nám to aj rozdeliť dáta na dôležité, ktoré majú napr. poputovať do dátového skladu na ďalšiu analýzu, a na menej dôležité, ktoré môžeme uložiť na disku pre

potrebu archivácie. Pri klasickom prístupe, keď sa všetky dáta najprv nahromadia, potom sa predspracujú a pošlú do dátového skladu na analýzu, nie je možné reagovať na možné negatívne udalosti v reálnom čase.

Klasický prístup k analýze dát v dátovom sklade:



Nový prístup analyzovania streamov:



Príklady na zdroje streamov

Pozrime sa na niekoľko príkladov, kde streamovaných dát stále pribúda. (7)

Dáta zo senzorov

Veľký rozmach internetu vecí zapríčiňuje, že stále sa inštalujú nové senzory a takmer o každej veci zbierame dáta. Tieto senzory produkujú neustály príval nových dát, ktoré musia byť hneď spracované.

Dáta zo sieťovej prevádzky

Switche a iné komponenty sieťovej prevádzky spracúvajú veľké množstvo IP packetov, ktoré prichádzajú v streamoch. V minulosti switche nepotrebovali spracovávať tieto dáta, len poslali ďalej packety. V súčasnosti ale dominuje trend pridávať do sieťových komponentov viacej inteligencie, aby sa dali detegovať počítačové útoky, ako napr. denial-of-service útoky.

Videá, fotky

Satelity neustále posielajú na Zem terabyte-y nových snímok každý deň, ktoré treba spracovať. Bezpečnostné kamery vyprodukujú záznamy síce v relatívne nízkom rozlíšení,

ale len v samotnom Londýne je viac ako 6 miliónov bezpečnostných kamier. Tieto dáta by mohli byť spracované na odhaľovanie zločinov v reálnom čase. Firmy chcú zistiť, ako sú zákazníci spokojní s ich produktmi aj na základe videí, ktoré zverejňujú na sociálnych sieťach.

Analyzovanie streamov

Pri analyzovaní streamov sa využívajú dva hlavné komponenty:

Stream: v tomto prípade dáta len dostaneme, hneď ich spracujeme a pošleme ďalej, teda sa nevyužíva žiadna pamäť, kde si zapamätáme predošlé výpočty. Takéto spracovanie dát je vhodné na filtrovanie dát, ako napr. bloom filtering, alebo na rozdelenie jedného streamu na viacej streamov napr. podľa typu dát, z ktorého senzoru prišli a pod. V tomto prípade ale nevieme robiť žiadne agregácie.

Window: asi najpoužívanejšia technika na spracovávanie streamov je technika posuvného okna. Zapamätáme si posledných n údajov, ktoré sme prijali v streame. Na toto okno sa môžeme pozerať ako na klasickú „statickú“ tabuľku v databáze. Teda nad touto tabuľkou sa môžeme dopytovať ako v tradičných relačných databázových systémoch, čo nám dáva veľké analytické možnosti. Treba si ale uvedomiť, že dopyty musia byť veľmi rýchle, lebo okno sa nám neustále mení. Sú aj iné možnosti, ako skonštruovať naše posuvné okno, najpoužívanejšie sú:

1) Zapamätáme si posledných n hodnôt. Toto sa volá posuvné okno, sliding window.
2) Iná variácia posuvného okna je, že si zapamätáme všetky hodnoty, ktoré prišli za posledných n sekúnd.
3) Hodnoty sa akumulujú v okne nejaký časový úsek a potom sa spracuje toto okno a všetky hodnoty sa vymažú z okna a celý proces sa začína odznova. Toto sa volá „skákajúce okno“, jumping window. Podobne ako pri posuvnom okne, aj tu môže byť podmienka zadaná vo forme nie časového úseku, ale počtu kusov riadkov.

Je možné skonštruovať aj iné komplikovanejšie podmienky, ktoré riadky majú zostať v danom okne.

1) Zapamätáme si posledných n hodnôt pre niektorý nekľúčový stĺpec. Typickým príkladom môže byť zapamätanie si n hodnôt pre každý senzor, alebo typ senzoru, ak ide o dáta z internetu vecí.
2) Predošlé prístupy vždy presne „odsekli“ tie dáta, ktoré už nepotrebujeme. Je ale možné, že chceme, aby sme brali do úvahy aj dáta z minulosti, ale aby tie zo súčasnosti mali väčšiu váhu. Využívajú sa tzv. decay windows, ktoré sú využívané napr. na hľadanie aktuálne najvyskytovanejšej hodnoty.

1 https://itpeernetwork.intel.com/sap-hana-intel-xeon-leadership/#gs.6uY2z6Qp
2 https://memcached.org/about
3 https://redis.io/topics/persistence
4 In memory data management (Prof. Hasso Plattner- Open HPI course)
5 https://blog.agilityworks.co.uk/our-blog/demystifying-the-column-store-store-statistics-in-sap-hana-and-the-benefits-for-business-in
6 https://docs.oracle.com/en/database/oracle/oracle-database/12.2/inmem/intro-to-in-memory-column-store.html#GUID-BFA53515-7643-41E5-A296-654AB4A9F9E7
7 Mining of Massive Datasets (Anand Rajaraman, Jure Leskovec, and Jeffrey D. Ullman, Standford)
8 The In-Memory Revolution: How SAP HANA Enables Business of the Future – Hasso Platner
9