Databázové technologie

Další databázové technologie

Petr Krajča

Katedra informatiky
Univerzita Palackého v Olomouci

Motivace

Výhody rel. databází

- ACID, konzistence dat
- normalizace dat, součást rel. dat. modelu, úzky vztah k logice, silná typovost
- možnost optimalizovat prováděné dotazy
- podpora komoditního hardware
- integrace s existujícím softwarem, víceméně uživatelsky přívětivé rozhraní

Problematické body rel. databází

- škálování, co
  - do velikosti dat (tera až petabyty dat)
  - i do velikosti požadavků (tisíce/sekundu)
- silná typovost
- rozhraní a normalizace dat se může míjet s potřebami aplikací

Limity škálování

- škálování pomocí clusteru volně vázaných počítačů limituje CAP teorém
- lze si vybrat maximálně dvě vlastnosti:
- **C**onsistency (konzistence) – každé čtení vrátí poslední zapsanou hodnotu, nebo chybu
- **A**vailability (dostupnost) – pro každý dotaz je vrácena nechybová odpověď
- **P**artition tolerance (odolnost k výpadku části) – systém funguje správně, pokud se zprávy zpozdí nebo ztratí

důsledek

- máme-li distribuovaný databázový systém
- v případě výpadku části (síťové komunikace) je volba:
  - obětovat dostupnost a zrušit operaci (operace) na úkor konzistence, nebo
  - obětovat konzistenci na úkor dostupnosti.

Dvoufázový potvrzovací protokol (1/2)

- protokol pro distribuované transakce
- předpoklady
  - každý uzel obsahuje write-ahead-log (WAL), ve kterém nejsou data nikdy poškozena
  - každé dva uzly spolu mohou komunikovat
  - žádný uzel nemá trvalý výpadek
- koordinátor + jednotlivé uzly

první fáze

1. koordinátor pošle všem uzlům dotaz (obsah transakce) a čeká na potvrzení od všech
2. uzly provedou transakci, ale nepotvrdí (neproběhne commit), změny jsou zapsány do undo a redo logů
3. každý uzel vrátí Ok/Fail podle toho, zda může transakci provést, nebo ne

Dvoufázový potvrzovací protokol (2/2)

druhá fáze

- commit (všechny uzly se shodly na provedení transakce)
  1. koordinátor pošle zprávu *commit* všem uzlům
  2. jednotlivé uzly potvrdí transakci (provedou změny) a pošlou potvrzovací zprávu koordinátorovi
  3. transakce je dokončena, pokud koordinátor obdržel potvrzovací zprávu od všech uzlů

- rollback (nebylo možné provést transakci minimálně na jednom z uzlů)
  1. koordinátor pošle zprávu *rollback* všem uzlům
  2. jednotlivé uzly zruší transakci (vrátí změny s pomocí undo logu) a pošlou potvrzovací zprávu koordinátorovi
  3. transakce je zrušena, pokud koordinátor obdržel potvrzovací zprávu od všech uzlů

poznámky

- prakticky používaný protokol
- blokující protokol, proto vhodný jen pro menší clustery

Map-reduce zpracování dat (1/4)

- cca 2006 Google popsal implementaci svého (tehdejšího) výpočetního clusteru
- cluster komiditního hardwaru
- vlastní framework zpracovávající data distribuovaným způsobem
- vstupem jsou:
  - data uložená v distribuovaném uložišti ve tvaru $\langle key, value \rangle$
  - funkce f, která jako vstup bere jednu dvojici $\langle k, v \rangle$ a vrací multimnožinu dvojic $\{\langle k_1, v_1 \rangle, \ldots, \langle k_n, v_n \rangle\}$
  - funkce g, která jako vstup bere multimnožinu dvojit $\{\langle k, v_1 \rangle, \ldots, \langle k, v_n \rangle\}$ (mají stejný klíč) a vrací hodnotu $\langle key, value \rangle$ (která vstupní dvojice agreguje)
- výstupem je multimnožina dvojic $\langle key, value \rangle$
- formát dat není limitující, jako klíč i hodnotu lze zákodovat složitější data

Map-reduce zpracování dat (2/4)

zpracování dat

1. data jsou načtena z uložiště případně upravena do vhodné podoby
2. **(fáze map)** na každou dvojici klíč-hodnota je aplikována funkce f, tj. je vytvořena multimnožina dvojic (fakticky operace flatMap nebo mapcan, umožňuje implementovat další operace, např. filter)
3. **(fáze reduce)** hodnoty vygenerované ve fázi map jsou uspořádány podle klíče a je na ně aplikovaná funkce g (podobnost s operacemi fold, groupBy)
4. výsledek předchozího kroku je uložen

poznámky

- obě operace snadno paralelizovatelné (distribuovatelné)
- mezi map a reduce je ještě vložena fáze combine, tj. operace reduce provedena na jednom uzlu
- programátor (operátor) dodá data + funkce, distribuci výpočtu, výpadky uzlů řeší framework
- Hadoop – otevřená implementace, dále různé varianty (optimalizované podle použití)

Map-reduce zpracování dat (3/4)

příklad výpočet četnosti písmen, slov, ...

- f funkce mající jako vstup řetězec a vracející multimnožinu dvojice $\langle znak, 1\rangle$
- g funkce, která sečte hodnoty v_i ve dvojicích $\{\langle k, v_1\rangle, \ldots, \langle k, v_n\rangle\}$

map

$f(alice) = \{\langle a,1\rangle, \langle l,1\rangle, \langle i,1\rangle, \langle c,1\rangle, \langle e,1\rangle\}$,
$f(barbara) = \{\langle b,1\rangle, \langle a,1\rangle, \langle r,1\rangle, \langle b,1\rangle, \langle a,1\rangle, \langle r,1\rangle, \langle a,1\rangle\}$,
$f(carol) = \{\langle c,1\rangle, \langle a,1\rangle, \langle r,1\rangle, \langle o,1\rangle, \langle l,1\rangle\}$.

reduce

$g(\{\langle a,1\rangle, \langle a,1\rangle, \langle a,1\rangle, \langle a,1\rangle, \langle a,1\rangle\}) = \langle a,5\rangle$,
$g(\{\langle b,1\rangle, \langle b,1\rangle\}) = \langle b,2\rangle$,
$g(\{\langle c,1\rangle, \langle c,1\rangle\}) = \langle c,2\rangle$,
$g(\{\langle e,1\rangle\}) = \langle e,1\rangle$,
$g(\{\langle l,1\rangle, \langle l,1\rangle\}) = \langle l,2\rangle$,
$g(\{\langle i,1\rangle\}) = \langle i,1\rangle$,
$g(\{\langle o,1\rangle\}) = \langle o,1\rangle$,
$g(\{\langle r,1\rangle, \langle r,1\rangle, \langle r,1\rangle\}) = \langle r,3\rangle$.

Map-reduce zpracování dat (4/4)

- úzkým hrdlem je synchronizace dat, ukládání dat mezi jednotlivými fázemi výpočtu
- v případě výpadku uzlu se lze vrátit k uloženým datům
- normálně pomalá operace, nota bene u distribuovaného uložiště

Apache Spark

- postaveno na RDD (Resilient distributed datasets)
- vstupní data jsou rozdělena na bloky, na ně jsou následně aplikovany operace typu map/reduce
- košatější API, předpokládá se více různých propojených fází (framework sestaví a optimulizuje DAG operací)
- eliminace ukládání dat na disk
- pro případ výpadku nejsou ukládána výsledná data, ale pouze odkaz na vstupní data (část RDD) a operace, které s daty mají být provedeny

NoSQL hnutí (1/3)

- nástup webových aplikací ovlivnil způsob práce s daty
- často semistrukturovaná data a jednoduché operace typu CRUD (create, read, update, delete)
- normalizace dat může být zbytečný a zdržující krok
- silná typovost zajišťuje kvalitu dat na úkor agilnosti vývoje
- nutnost zajišťovat přístupnost pro velké množství současně připojených uživatelů

charakteristické rysy

- key-value uložiště (různého typu)
- API postavané na HTTP, JSON, JavaScriptu
- zpracování dat metodou map-reduce

NoSQL hnutí (2/3)

sloupcové databáze

- nebo též sloupcově orientované databáze
- data jsou ukládána po sloupcích (vs. tradiční databáze ukládající jednotlivé řádky)
- vhodné pro analytické dotazy, kdy potřebujeme pracovat jen s vybranými sloupci
- data jsou v blocích homogenní (snadná komprimace, snížení I/O operací)
- buď jako specializované SŘBD – MonetDB, ClickHouse, Apache Druid
- součást relačních SŘBD – PostgreSQL a cstore, MariaDB a ColumnStore

key-value uložiště

- zjednodušení struktury dat i dotazů zjednoušuje práci s daty, usnadnění distribuce dat mezi více uzlů
- cílem je maximalizovat rychlost a dostupnost (různé struktury)
- často jako cache, součást většího celku
- BerkeleyDB, Redis a Memcached (in-memory uložiště s možností uložení na disk), Amazon Dynamo, Riak

NoSQL hnutí (3/3)

dokumentové databáze

- specifický případ key-value databáze
- semistrukturovaná data
- hodnoty navenek reprezentovány jako JSON nebo XML dokument
- je možné pracovat s daty uvnitř dokumentu
- RESTful API
- podobnou funkcionalitu lze získat v tradičních SŘBD pomocí specializovaných datových typů (xml, json)

grafové databáze

- v případě relačních, key-value i dokumentových databází jsou vztahy mezi daty vyjádřeny převážně implicitně
- tento problém řeší grafové databáze (Neo4J)
- dále se se dá setkat s tzv. RDF databázemi (někdy též triplestore)
- data jsou reprezentována jako trojice předmět-predikát-objekt, např. *,,Paul McCartney" – ,,byl členem" – ,,The Beatles"*
- sémantické dotazy, jazyk SPARQL