

NoSQL databáze

Marek Rychlý (a Dušan Kolář)

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů

Přednáška pro PDB
15. října 2013



- 1 Od relačních databází k NoSQL
 - Proč nestačí relační databáze?
 - Co jsou NoSQL databáze?
 - Co přináší NoSQL databáze?
- 2 Příklady NoSQL databází
 - mongoDB
 - Oracle NoSQL
- 3 Shrnutí a závěr



Relační databáze

- Data organizována do tabulek, řádek reprezentuje záznam.
(koncept matematické relace, řádek prvkem relace nad doménami sloupců tabulky)
- Každý sloupec má přesně daný (jednoduchý) datový typ.
(tj. množina/doména odpovídající části relace)
- Záznam v tabulce se může odkazovat na záznam (jiné) tabulky.
(hodnota cizího klíče odpovídá hodnotě primárního klíče odkazovaného záznamu)
- Organizace dat musí splňovat normální formy.
(1NF, 2NF, 3NF, EKNF, BCNF, 4NF, 5NF, DKNF, 6NF¹; jinak hrozí redundance/chyby)
- Dotazy a úpravy nad daty pomocí SQL.
(dotazování pomocí SELECT vychází z relační algebry)
- Databázový systém zaručuje ACID změn uložených dat.
(Atomicity, Consistency, Isolation, Durability)

¹EKNF = Elementary Key NF; DKNF = Domain/key NF



ACID

Atomicity Atomičnost transakcí, žádný rozpracovaný stav a to i ve vztahu k možné chybě OS či HW.
(proběhne celá transakce, tj. všechny její změny, nebo nic)

Consistency V DB jsou pouze platná data dle daných pravidel. Transakce se neuskuteční, pokud to nelze dodržet, jinak platí, že původní i nový stav je platný.

Isolation Souběžné transakce se neovlivňují. Serializace. Pořadí však není zajištěno.

Durability Uskutečněná transakce nebude ztracena (její projev). Podpora obnovy dat po pádu HW/SW.

ACID netriviální, omezuje změny dat (zamykání) a přístup k datům (rychlost).



Požadavky na moderní databáze

- cloud, distribuované databáze
(decentralizace úložiště dat, úmyslná redundance pro odolnost proti výpadkům a rychlost, velké objemy dat a velké množství operací /big data/, atd.)
- problematické datové typy
(údaje klíč-hodnota, objekty, nestrukturované dokumenty, RDF grafy, atp.)
- iterativní vývoj
(časté změny schématu databáze nebo dokonce žádné schéma, různé/nejasné způsoby použití databáze, atp.)
- vysoké požadavky na škálovatelnost
(mobilní zařízení jako klienti i úložiště/poskytovatelé dat, nerovnoměrné rozložení zátěže prostorově i časově, specifické požadavky na dostupnost, předem neznámé dotazy nelze optimalizovat indexy, atp.)



Moderní relační databáze?

- Snaha přizpůsobit relační databázi moderním požadavkům.
(post-relační relační databáze /objektově-relační, s podporou XML, . . . /, univerzální datové modely, úmyslná denormalizace, zavádění cache, datové sklady, atd.)
- „Relační“ databáze přestává odpovídat relačnímu konceptu.
(už ne matematické relace, ale spíše kolekce/množiny/grafy nestruturovaných dat)
- Dodržování ACID nevhodně omezuje práci s databází.
(úmyslné zanedbání/odpuštění Atomicity, Consistency, Isolation nebo Durability pro získání rychlosti a dostupnosti dat)
- Vznik specializovaných nerelačních (post-relačních) databází:
 - pro specificky strukturovaná data,
(čistě objektové či XML databáze, úložiště tagovaných dokumentů, atp.)
 - pro specificky uložená/přístupovaná data.
(in-memory, embedded a real-time databáze či map-reduce zpracování dat)



Teorém CAP u sdílených/distribuovaných systémů

Consistency

každý uzel/klient vidí ve stejný čas stejná data,
(data konzistentní nezávisle na běžících operacích či jejich umístění)

Availability

každý požadavek obslužen, úspěšně nebo neúspěšně,
(nepřetržitý provoz, vždy možnost zapsat a číst data)

Partition Tolerance

funkční navzdory chybám sítě nebo výpadkům uzlů.
(možnost výpadku části infrastruktury, např. odstávka pro údržbu)

Teorém

U sdílených systémů možné uspokojit maximálně 2 ze 3 požadavků...
Eric Brewer (+ N. Lynch), 2000



CA / CP / AP

- **Consistence + Availability**
= 2fázový commit, protokoly pro (in)validaci cache
(např. Cluster databases, LDAP, xFS file system)
- **Consistency + Partition tolerance**
= agresivní zamykání, ustojí malé výpadky
(např. distribuované db. a zamykání, protokol pro většinovou shodu)
- **Availability + Partition tolerance**
= střídání uzlů, řešení konfliktů, optimistická strategie
(Coda, Web caching[*sic!*], DNS)

Brewer, Eric A.: Towards Robust Distributed Systems. Portland, Oregon, July 2000. Keynote at the ACM Symposium on Principles of Distributed Computing (PODC) on 2000-07-19.



NoSQL („Not only SQL“)

- NoSQL podporují nerelační datový model.
(klíč-hodnota, dokumentové, grafové, atd.)
- NoSQL podporují distribuovanou architekturu.
(lze použít jako centrální db., ale jejich síla je v distribuovanosti)
- Většina NoSQL je open-source,
mají různý přístup k práci s daty a jejich dotazování.
- NoSQL většinou řeší CAP omezením konzistence dat.
(BASE = Basically Available Soft-state services with Eventual-consistency)



BASE

Basically Available

An application works basically all the time . . .

Soft-state

. . . it does not have to be consistent all the time . . .

Eventual consistency

. . . but it will be in some known-state state eventually.

Christof Strauch: NoSQL Databases, Hochschule der Medien, Stuttgart.

Případné nekonzistence jsou řešeny při čtení (např. verzování, nevalidní cache), při zápisu (např. distribuce změn), nebo asynchronně (např. replikace dat).



ACID vs. BASE

ACID:

- silná konzistence
- izolovanost
- orientace na komit
- vnořené transakce
- dostupnost?
- konzervativní (pesimistické)
- složitá evoluce (schématu, ...)

BASE:

- slabá konzistence (stará data)
- dostupnost na prvním místě
- přibližné odpovědi jsou OK
- jednodušší, rychlejší
- dodávka dat „jak to jen půjde“
- agresivní (optimistické)
- jednodušší evoluce



NoSQL databáze klíč-hodnota

- Jeden klíč, jedna hodnota, žádný duplikát.
(klíč může být složený, např. z hlavní a upřesňující části, které lze použít jako ID struktury a ID její položky)
- Přístup podle klíče přes hash tabulky
(brutálně rychlé)
- Hodnota je BLOB, databáze se to ani nesnaží chápat.
(zpracování obsahu „hodnoty“ je na aplikaci, databáze ji jen uchovává jako celek)
- Pokud nás zajímá jen část hodnoty, ať pro dotazy, nebo pro zápis, tak je poměrně neefektivní.
(lze řešit vyjmutí části pod záznam s vlastním „klíčem“, např. s upřesňující částí)

Např. Oracle NoSQL, Dynamo (by Amazon)



NoSQL dokumentové databáze

- V podstatě „klíč-hodnota“, ale hodnota je strukturovaná.
(databáze vidí „dovnitř“, hodnota je pochopena, analyzována)
- Hodnota např. jako XML/JSON, nebo jako objekt.
(možnost referenční na jiné záznamy, vnořování struktur, kolekce)
- Dotazy i složitější, než přes klíče.
(např. XPath nebo jako v objektových databázích)

Např. mongoDB



Sloupcové NoSQL databáze

- Řádky jako v RDB, u řádku máme různé sloupce s hodnotami.
(tj. u řádku je kolekce klíč-hodnota dvojic, kde „klíč“ je název sloupce; sloupce mohou být pro každý řádek různé)
- Můžeme mít adresáře (supercolumn).
(pak řádek obsahuje kolekci supersloupců, z nichž každý obsahuje kolekci slouců)
- Řídká, vícedimenzionální, uspořádaná mapovací funkce.
(řádky \times sloupce, ale struktura řádku není dána, každý může mít různé sloupce)

Např. Cassandra (by Facebook), BigTable (by Google)



Grafové NoSQL databáze

- Grafy = uzly, vlastnosti uzlů, hrany spojující uzly.
- Různé implementace úložiště.
(nastavitelné, generické, uživatelovo)
- Použití pro reprezentaci sítí a jejich topologií.
(např. sociální či dopravní sítě, topologie počítačových sítí, ...)
- RDF databáze jsou specifickou kategorií grafových NoSQL.
 - RDF je orientovaný ohodnocený graf, kde hrana začíná v „subjektu“, je ohodnocena „predikátem“ a končí v „předmětu“.
 - Subjekt a predikát jsou reprezentovány URI.
 - Předmět (object) je hodnota nebo URI odkazující na nějaký předmět.
 - Nad RDF grafem je možno dokazovat fakta.
(např. pokud platí predikát na subjektu a předmětu, pak ...)
 - Standardizovaný odtazovací jazyk SPARQL.

Např. Neo4j, AllegroGraph (RDF)



Relační databáze vs. NoSQL databáze

- NoSQL databáze jsou moderní, oblíbené zejména v cloudu.
- Nicméně NoSQL jsou vhodné jen pro specifické případy.
(distribuovaná úložiště či zpracování, rychlost na úkor ACID)
- Pro klasické informační systémy stále nejlepší relační db.
- Při výběru NoSQL potřeba zvažovat
 - druh úložiště, organizace dat,
(klíč-hodnota, dokumentové, sloupcové, grafové, ...)
 - vlastnosti distribuované architektury,
(s/bez koordinátorem, výpadky uzlů a koordinátora /konzensus/, atp.)
 - možnosti škálovatelnosti,
(směrem k distribuovanosti/rozsáhlosti i k ACID bezpečnosti)
 - možnosti integrace do aplikace.
(použití s MapReduce/Apache Hadoop, „inteligentní“ drivery, ...)
- Rychlost, výkon, atp. posuzovat až nakonec, stále se vyvíjí.



mongoDB

- **Multiplatformní open-source dokumentová NoSQL databáze.**
(implementována v C++, dostupná pro Windows, Linux, MacOS X a Solaris)
- **Dlouhý vývoj, rozšířená, první vydání v roce 2009.**
(používají Craigslist, eBay, Foursquare, SourceForge, The New York Times, ...)
- `http://www.mongodb.org/`
(manuální, instalační balíčky, demo s interaktivním shell-em)
- **Dokument = JSON object, PHP array, Python dict, Ruby hash,**
(podpora mnoha ovladačů resp. programovacích jazyků klientských aplikací)



Normalizace a de-normalizace



relační schéma



schéma dokumentu

(diagramy převzaty z „Jeremy Mikola: Schema Design. MongoDB, 2013-10-11.“)



Vnořené dokumenty, reference, dědičnost

- Dokumenty je možno vnořovat, ale také odkazovat přes jejich ID.
(vizte vnořené kolekce Comment, Tag, Category a odkazovaný User)
- Vnořováním se typicky realizuje vztah 1:N (agregace, kompozice).
(vnořený dokument pak neoddělitelně patří svému nadřazenému dokumentu)
- Odkazováním se realizuje vztah 1:N a N:M (asociace).
(reference lze vést z obou stran vazby 1:N)
 - 1:N lze mít v odkazujícím kolekci N odkazovaných,
(vhodné, pokud je N konečné a nevelké číslo)
 - N:1 lze mít referenci na 1 odkazovaného u každého z N odkazujících,
(vhodné, pokud může být N neomezené)
 - N:M jsou kolekce odkazů na obou stranách vazby.
- Pomocí referencí lze vytvářet stromy/grafy dokumentů.
(rodičovský uzel odkazuje na potomky, nebo potomci odkazují na rodiče)
- Dokumenty je možno dědit/odvozovat a poté rozšiřovat.



Ukázka práce s mongoDB

```
db.article.insert({
  "name" : "My Article",
  "publish date" : new Date("2013-10-15"),
  "comment" : [],
  "tag" : [ "adventure", "fiction" ]
})

db.article.find({"tag" : "adventure"}).pretty()
{
  "_id" : ObjectID("525c7ed0cc9374393401f5fd"),
  "name" : "My Article",
  "publish date" : ISODate("2013-10-15"),
  "comment" : [],
  "tag" : [
    "adventure",
    "fiction"
  ]
}

db.article.update(
  {"_id" : new ObjectID("525c7ed0cc9374393401f5fd")},
  {$push : { comment : { name : "Alice", comment : "Awesome post!" } } } )
```

Pozn.: „\$push“ je atomický update operátor (další jsou \$pop, \$pull, ...).



Oracle NoSQL

- Multiplatformní closed-source klíč-hodnota NoSQL databáze.
(impl. v Java, poskytována v Community Edition/AGPL3 nebo Enterprise Edition)
- Využívá (Oracle) Berkeley DB, jedna prvních klíč-hodnota db.
- Dobrá integrace s ostatními (nejen) Oracle produkty.
(Oracle Database, Oracle Fusion Middleware, Apache Hadoop, ...)
- `http://www.oracle.com/goto/nosql`
(manuální, instalační balíčky, ...)
- Dvousložkový String klíč (major, minor) a netypovaná hodnota.
(záznamky se stejným major-klíčem musí být na stejném uzlu/lokaci)
- Podpora ACID, ale pouze u záznamů se stejným major-klíčem.
(díky shodnému umístění těchto záznamů odpadají problémy s CAP teorémem)
- API a Java ovladač směřující dotazy na správný uzel.
(na server mající daná data v distribuované db.; API pro CRUD operace a iterátory)



Ukázka CRUD operací a iterátoru v Oracle NoSQL

```
// Put a new key/value pair in the database, if key not already present.
Key key = Key.createKey("Katana");
String valString = "sword";
store.putIfAbsent(key, Value.createValue(valString.getBytes()));

// Read the value back from the database.
ValueVersion retValue = store.get(key);

// Update this item, only if the current version matches the version I read.
// In conjunction with the previous get, this implements a read-modify-write
String newvalString = "Really_nice_sword";
Value newval = Value.createValue(newvalString.getBytes());
store.putIfVersion(key, newval, retValue.getVersion());

// Create Iterator and iterate over the store.
Iterator<KeyValueVersion> iter = store.storeIterator(Direction.UNORDERED, 100);
while (iter.hasNext()) {
    KeyValueVersion keyVW = iter.next();
    Value val = keyVW.getValue();
    Key key = keyVW.getKey();
    System.out.println(val.toString() + "_" + key.toString() + "\n");
}

// Finally, (unconditionally) delete this key/value pair from the database.
store.delete(key);
```



Ukázka CRUD operací v transakci

```
// Create a sequence of operations.
OperationFactory of = store.getOperationFactory();
List<Operation> opList = new ArrayList<Operation>();

// Create major and minor path components.
List<String> majorComponents = new ArrayList<String>();
List<String> minorLength = new ArrayList<String>();
List<String> minorYear = new ArrayList<String>();
majorComponents.add("Katana");
minorLength.add("length");
minorYear.add("year");
Key key1 = Key.createKey(majorComponents, minorLength);
Key key2 = Key.createKey(majorComponents, minorYear);

// Now put operations in an opList.
String lenVal = "37";
String yearVal = "1454";
opList.add(of.createPut(key1, Value.createValue(lenVal.getBytes())));
opList.add(of.createPut(key2, Value.createValue(yearVal.getBytes())));

// Now execute the operation list.
store.execute(opList);
```



Shrnutí a závěr

- NoSQL databáze určeny pro uchování a práci s nerelačními daty v distribuovaných prostředích.
(NoSQL nenahrazují relační databáze)
- Namísto ACID poskytují BASE², což nese svá specifika.
(nemohou současně poskytnout vše z Consistency, Availability, Partition Tolerance)
- Různé typy NoSQL databází dle způsobu uložení dat.
(klíč-hodnota, dokumentové, sloupcové, grafové, ...)

²BASE = Basically Available, Soft-state, Eventual consistency

