Databázové systémy

NoSQL

Literatúra

Pramod J. Sadalage and Martin Fowler: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, 2012 by Addison-Wesley Professional.

Príklad

- Simple Web Analytics
  - id, user_id, url, pageviews

Príklad

- Simple Web Analytics
  - id, user_id, url, pageviews
- A zrazu to začne byť úspešné...

Príklad

- Simple Web Analytics
  - id, user_id, url, pageviews
- A zrazu to začne byť úspešné...
  - a začnú prichádzať timeouts pri insertovaní

Príklad

- Simple Web Analytics
  - id, user_id, url, pageviews
- A zrazu to začne byť úspešné...
  - a začnú prichádzať timeouts pri insertovaní
  - Riešenie: dávkové spracovanie, pridáte queue a workera, ktorý inserte vždy 1000 záznamov naraz

Príklad

- Simple Web Analytics
  - id, user_id, url, pageviews
- A zrazu to začne byť úspešné...
- A ešte viac úspešné
  - Worker nestíha, fronta rastie
  - Viac workerov, databáza je bottleneck
  - Riešenie: sharding (horizontal partitioning)
    - Musím prepísať skoro celú svoju aplikáciu (backend)

Príklad

- Simple Web Analytics
  - id, user_id, url, pageviews
- A zrazu to začne byť úspešné...
- A ešte viac úspešné
- A ešte...
  - Zistím, že som zvolil málo shardov
  - Resharding je peklo...

Príklad

- Simple Web Analytics
  - id, user_id, url, pageviews
- A zrazu to začne byť úspešné...
- A ešte viac úspešné
- A ešte...
- A už mám tak veľa hardvéru, že mi na jednom stroji odíde disk
  - Nechávam v “pending” queue, master/slave replikácia

NoSQL

- Objavujú sa problémy v súvislosti s manažmentom a analýzou dát, pre ktoré nie sú prístupy založené na relačných databázach najvhodnejšie
- NoSQL tu vo význame “nie RDBMS”

NoSQL

- Objavujú sa problémy v súvislosti s manažmentom a analýzou dát, pre ktoré nie sú prístupy založené na relačných databázach najvhodnejšie
- NoSQL tu vo význame “nie RDBMS”
- Neznamená to, že sa nesmie používať SQL
  - existujú NoSQL prístupy využívajúce SQL :)

Čo nám poskytuje DBMS?

- efektívne
- spoľahlivé
- vhodné
- bezpečné
- viac-používateľske
- ukladanie a prístup k veľkému množstvu perzistentných dát

Niekedy až príliš veľký balík

- efektívne
- spoľahlivé
- vhodné
- bezpečné
- viac-používateľske
- ukladanie a prístup k veľkému množstvu perzistentných dát

Niekedy až príliš veľký balík

- efektívne
- spoľahlivé
- vhodné
- bezpečné
- viac-používateľske
- ukladanie a prístup k veľkému množstvu perzistentných dát

Čo robí RDBMS “convenient”

- Jednoduchý dátový model
- deklaratívny dopytovací jazyk
- transakcie, obmedzenia

Čo robí RDBMS “convenient”

- Jednoduchý dátový model
- deklaratívny dopytovací jazyk
- transakcie, obmedzenia
- všetko to spolu hrá v jednom balíku

Čo robí RDBMS “convenient”

- Jednoduchý dátový model
- deklaratívny dopytovací jazyk
- transakcie, obmedzenia
- všetko to spolu hrá v jednom balíku
  - Je to +/- štandardizované, všetci to používajú a chápu

Čo robí RDBMS “convenient”

- Jednoduchý dátový model
- deklaratívny dopytovací jazyk
- transakcie, obmedzenia
- všetko to spolu hrá v jednom balíku
  - Je to +/- štandardizované, všetci to používajú a chápu
    - V minulosti DB často ako **integračná db**

Problémy

- Impedance mismatch
  - Rozdiel medzi dátovými štruktúrami v pamäti vs. relačný model
  - Tuple vs. nested records, lists, hashes...

Problémy

- Impedance mismatch
  - Rozdiel medzi dátovými štruktúrami v pamäti vs. relačný model
  - Tuple vs. nested records, lists, hashes...
- ORM?
  - Ak si nebudem databázu všímať, ona sa mi svojim výkonom raz pripomenie

Problémy

- Impedance mismatch
  - Rozdiel medzi dátovými štruktúrami v pamäti vs. relačný model
  - Tuple vs. nested records, lists, hashes...
- Integračná databáza → aplikačná databáza
  - Textové API cez HTTP protokol (XML, **JSON**)
    - A toto je rich data structure, nemusím prenášať relácie

Problémy

- Impedance mismatch
  - Rozdiel medzi dátovými štruktúrami v pamäti vs. relačný model
  - Tuple vs. nested records, lists, hashes...
- Integračná databáza → aplikačná databáza
  - Textové API cez HTTP protokol (XML, **JSON**)
    - A toto je rich data structure, nemusím prenášať relácie
- Fungovanie v clustri
  - Rozmach webu a dát, ktoré sa vygenerujú z trafficu

Problémy

- Impedance mismatch
  - Rozdiel medzi dátovými štruktúrami v pamäti vs. relačný model
  - Tuple vs. nested records, lists, hashes...
- Integračná databáza → aplikačná databáza
  - Textové API cez HTTP protokol (XML, **JSON**)
    - A toto je rich data structure, nemusím prenášať relácie
- Fungovanie v clustri
  - Rozmach webu a dát, ktoré sa vygenerujú z trafficu
  - A relačné db neboli navrhnuté na beh v clustri

Čo robí RDBMS “convenient”

- Jednoduchý dátový model
- deklaratívny dopytovací jazyk
- transakcie
- všetko to spolu hrá v jednom balíku
  - problém je, ak nám k tomu nehrajú dáta

kedy to nie je až tak “convenient”

- keď máme nerelačné dáta
  - veľký preprocessing na to, aby sme dáta dostali do tabuliek
- keď nepotrebujeme komplikované dopyty
  - možno nám stačí jednoduchý fetch (key-value)
- keď nepotrebujeme dokonalú garanciu transakcií a totálnu fail-proof
  - ale radšej by sme to mali celé rýchlejšie

Iné požiadavky aplikácií

- jednoduchší (resp. nerelačný) model
- spoľahlivosť nie je kritická – vieme dať redo
- perzistencia – jednoduché súbory sú OK

- ukladanie a prístup k **veľkému množstvu** perzistentných dát
- Vysoká efektívnosť

Iné požiadavky aplikácií

- jednoduchší (resp. nerelačný) model
- spoľahlivosť nie je kritická – vieme dať redo
- perzistencia – jednoduché súbory sú OK

Obetujeme niečo za niečo

- ukladanie a prístup k **veľkému množstvu** perzistentných dát
- Vysoká efektívnosť

Viete čo je to cap?

?

?


VELKOPOPOVICKÝ
Kozel
zal. 1874

Asi treba ísť do angličtiny

?

Brewer's CAP Theorem

- Consistency
- Availability
- Partition Tolerance

Brewer's CAP Theorem

- Consistency
  - dvaja zákazníci si naraz nekúpia poslednú letenku
  - riešia databázy so svojim ACID (pamätáte si ešte?)
- Availability
- Partition Tolerance

Brewer's CAP Theorem

- Consistency
- Availability
  - služba je dostupná
  - Amazon: desatina sekundy navyše v response znamená 1% tržieb
  - Google: pol sekundy latencie navyše zníži traffic o päťinu
- Partition Tolerance

Brewer's CAP Theorem

- Consistency
- Availability
- Partition Tolerance
  - nastáva v prípade, že máme viacero serverov
  - partície vzniknú ak sa preruší spojenie medzi serverom A a serverom B
  - Partition Tolerance hovorí, že systém sa musí správať korektne až kým nenastane kompletný kolaps siete

Brewer's CAP Theorem

- Keď škálovať, tak horizontálne
  - scale out namiesto scale up
- Ale keď pôjde do tuhého, tak Vaša aplikácia CAP nedosiahne
  - môžete mať ľubovoľné dve z CAP, ale nie všetky tri
  - biznis hovorí, že je treba obetovať konzistenciu
    - Eventual consistency

Späť(?) k NoSQL

- Flexibilná schéma
- Jednoduchšie/lacnejšie nasetupovať (?)
- Masívna škálovateľnosť
  - veľa dát
  - bleskové spracovanie
  - veľa dát
- Nižšia konzistencia → vyšší výkon a dostupnosť

Nejaké nevýhody?

- Absencia deklaratívneho dopytovania
  - viac programovania
- Nižšia konzistencia
  - nižšia spoľahlivosť, resp. garancia dát

Príklad – web log

- CSV: UserID, URL, timestamp, additional-info
- Nahranie dát do RDB
  - čistenie dát
  - extrakcia dát
  - verifikácia
  - špecifikácia schémy

Príklad – web log

- CSV: UserID, URL, timestamp, additional-info
- Nahranie dát do RDB
  - ~~čistenie dát~~
  - ~~extrakcia dát~~
  - ~~verifikácia~~
  - ~~špecifikácia schémy~~
- NoSQL prístup: nerobte nič z toho, pracujte priamo s dátami zo súborov, kde sú uložené

Príklad – web log

- CSV: UserID, URL, timestamp, additional-info
- Dopytovanie: nájdi všetky záznamy
  - pre dané UserID
  - pre danú URL
  - pre tento timestamp
  - ktoré majú v additional-info niečo špeciálne

Príklad – web log

- CSV: UserID, URL, timestamp, additional-info
- Dopytovanie: nájdi všetky záznamy
  - pre dané UserID
  - pre danú URL
  - pre tento timestamp
  - ktoré majú v additional-info niečo špeciálne
- Nič z toho nevyžaduje SQL
- Ale je to paralelizovateľné

Príklad – web log

- CSV: UserID, URL, timestamp, additional-info
- Dopytovanie: nájdi páry UserID, ktoré pristupujú na rovnaku URL

Príklad – web log

- CSV: UserID, URL, timestamp, additional-info
- Dopytovanie: nájdi páry UserID, ktoré pristupujú na rovnaku URL
  - potrebujeme self-join

Príklad – web log

- CSV: UserID, URL, timestamp, additional-info
- Dopytovanie: nájdi páry UserID, ktoré pristupujú na rovnaku URL
  - potrebujeme self-join
  - ale úprimne...je to divná query, ktorú asi v aplikácii nebudete potrebovať :)

Príklad – web log

- CSV: UserID, URL, timestamp, additional-info
- CSV: UserID, name, age, gender
- Dopytovanie: nájdi priemerný vek používateľov pristupujúcich na danú URL
  - na toto je SQL fajn
  - ale ako je to s konzistenciou?

Príklad - wikipedia

- Veľká zbierka dokumentov
- kombinácia štrukturovaných a neštrukturovaných dát
- Nájdi úvodný paragraf všetkych stránok o amerických prezidentoch pred rokom 1900

Príklad - wikipedia

- Veľká zbierka dokumentov
- kombinácia štrukturovaných a neštrukturovaných dát
- Nájdi úvodný paragraf všetkych stránok o amerických prezidentoch pred rokom 1900

- Ako na toto vymyslieť relačnú schému?
- Konzistencia asi nie je až taká kritická
  - veď nejaká stránka možno ešte ani neexistuje

Polyglot persistence

- Časti našej aplikácie majú rôzne požiadavky
  - Finančné dáta – bezpečne schované za ACIDom relačných databáz
  - Nákupný košík v e-shope – v niečom rýchlom, ak tieto dáta neprežijú nejakú poruchu, tak my to prežijeme

NoSQL a schemaless-ness

- Väčšina NoSQL riešení nepotrebuje explicitne definovať schému dát
  - Niektoré to však podporujú (elasticsearch mapping)
- Každý záznam môže mať teoreticky úplne custom štruktúru
  - Ale asi by sa s tým v aplikácii ťažko pracovalo
- Schéma je teda prítomná implicitne v kóde
- ...a čo s migráciami?

S čím pracujú NoSQL databázy?

- V RDB máme záznamy rozbité do množstva riadkov v rôznych tabuľkách
- V NoSQL máme **agregáty**
  - Môže byť komplexná dátová štruktúra
    - S listami, podštruktúrami
  - S ktorou chceme v našich scenároch pracovať ako s jednou jednotkou
    - A na nej mať konzistenciu, atomicitu
  - Perfektné pre horizontálne škálovanie

Problémy agregátov

- Viacero možností ako definovať agregáty

users

{

“id” : 1,

name”: “Martin”,

“billingAddress”: [{city: Chicago}]

}

orders

{

“id” : 99,

“customer_id” : 1,

“orderItems”: [{“productId”: 27, “price”: 35.47,
“productName”: “NoSQL Distilled” },{...}],

“shippingAddress” : {“city”: {…}},

“paymentInfo”: {…}

}

customers with orders


```
{
“id” : 1,
name”: “Martin”,
“billingAddress”: [{city: Chicago}],
orders” : [{
    “id” : 99,
    “customer_id” : 1,
    “orderItems”: [{“productId”: 27, “price”: 35.47, “productName”: “NoSQL Distilled” },{...}],
    “shippingAddress” : {“city”: {…}},
    “paymentInfo”: {…}
}]
}
```

Problémy agregátov

- Viacero možností ako definovať agregáty
- Už pri návrhu schémy musím rozmýšľať nad tým, aké dopyty budem vykonávať a ako chcem, aby sa vykonávali
  - Pohľad zákazníka vs. pohľad manažéra, ktorý chce GROUP BY
    - To pri SQL nemusím riešiť

Agregáty a ACID

- Atomicita na úrovni agregátu
- Ak potrebujem atomickú operáciu cez viacero agregátov, tak si ju musím zabezpečiť na aplikačnej úrovni

..a keď už sme znova pri tom škálovaní

- Sharding
  - Rozdelenie dát na viacero serverov
  - Napr. objednávky z Európy v Írskom datacentre, z Nového Zélandu v Austrálskom
- Replikácia
  - Master-slave
  - peer-to-peer

Typy NoSQL systémov

- MapReduce framework
- Key-value storage
- Column storage
- Document storage
- grafové databázy


Redis
HBASE
Cassandra
mongoDB
riak
CouchDB
relax
Tokyo
Cabinet
HIVE
Scalaris
Project Voldemort
Neo4j
the graph database
membase