NoSQL

Přednáška č. 3

RDB

NoSQL databáze

- NoSQL == Not only SQL database

Distribuované databáze

- Distribuovaný systém je složen z několika počítačů (uzlů) a softwarových komponent, které spolu komunikují pomocí počítačové sítě.
- Distribuovaný systém může být založen na různých konfiguracích Mainframe, pracovní stanice, atd.
- Počítače pak spolu komunikují a sdílí zdroje systému, aby dosáhly společného cíle

Výhody DD

- **Spolehlivost (fault tolerance) :**
  Pokud některý uzel má výpadek, tak to neohrozí funkci celého systému.
- **Rozšiřitelnost (Scalability):**
  Distribuovaný systém je možné snadno rozšířit přidáním nových serverů atd.
- **Sdílení zdrojů (Sharing of Resources):**
  Sdílení je základní pro mnoho aplikací (např. bankovnictví, rezervační systémy atd.) Data nebo zdroje jsou sdíleny v distribuovaném systému (např. i tiskárny a jiné zdroje).
- **Flexibilita (Flexibility):**
  Díky své flexibilitě je snadné instalovat, implementovat a odladit nové služby.
- **Rychlost (Speed):**
  Distribuované systémy mohou mít daleko větší výkon než „normální“ systémy.
- **Otevřené systémy (Open system):**
  Je to otevřený systém, každá služba je rovnocenně dostupná pro každého klienta (lokálního i vzdáleného).
- **Výkon (Performance):**
  Kolekce procesorů v systému poskytuje vyšší výkon (lepší poměr výkon/cena) než centralizované systémy.

Nevýhody DD

- Hledání problémů, ladění
- Menší podpora software
- Síťová komunikace
  - Komunikace po síti může být problém (ztráta spojení, ztracené zprávy atd.).
- Bezpečnost

Škálovatelnost, rozšiřitelnost

- Schopnost přidat vlastnosti, aby byly uspokojeny nové nároky uživatelů (zákazníků)
- Např. u webu by to znamenalo upravit jej tak, aby jej mohlo využívat více lidí (např. přidání nového HW)
- **Vertikální škálování**
  - Přidat zdroje do již existujícího systému, aby se zvýšila jeho kapacita, průchodnost atd.
  - Např. přidat procesor, paměť či HDD
- **Horizontální škálování**
  - Přidat nové uzly (např. nový počítač do již existující infrastruktury)

Co je NoSQL

- NoSQL je nerelační systém řízení báze dat
- Navržen pro distribuovaná datová uložiště (např. Google, Facebook).
- Nevyžadují pevné schéma databáze
- Nepoužívá se JOIN operace
- Škálují horizontálně

Proč NoSQL

- Dostupnost dat je dnes o mnoho větší než byla v minulosti
- Osobní data uživatelů, sociální vazby (grafy), geografická data, logy systémů jsou příklady kdy množství dat roste exponenciálně.
- SQL databáze nebyly navrženy na zacházení s tak obrovským množstvím dat

SŘBD vs. NoSQL

- **SŘBD**
  - Strukturovaná a organizovaná data
  - SQL
  - Data a jejich vztahy jsou uloženy v oddělených tabulkách.
  - DML a DDL
  - Konzistence
  - Transakce ACID

- **NoSQL**
  - Nemá deklarativní dotazovací jazyk
  - Nemá definované schéma
  - Ukládá dvojice Klíč-hodnota
  - Uložení sloupců, dokumentů, grafů
  - Případná konzistence upřednostněna před ACID vlastnostmi
  - Podpora nestrukturovaných a nepředvídatelných dat
  - CAP teorém
  - Upřednostňují vysoký výkon, dostupnost a rozšiřitelnost

Historie NoSQL

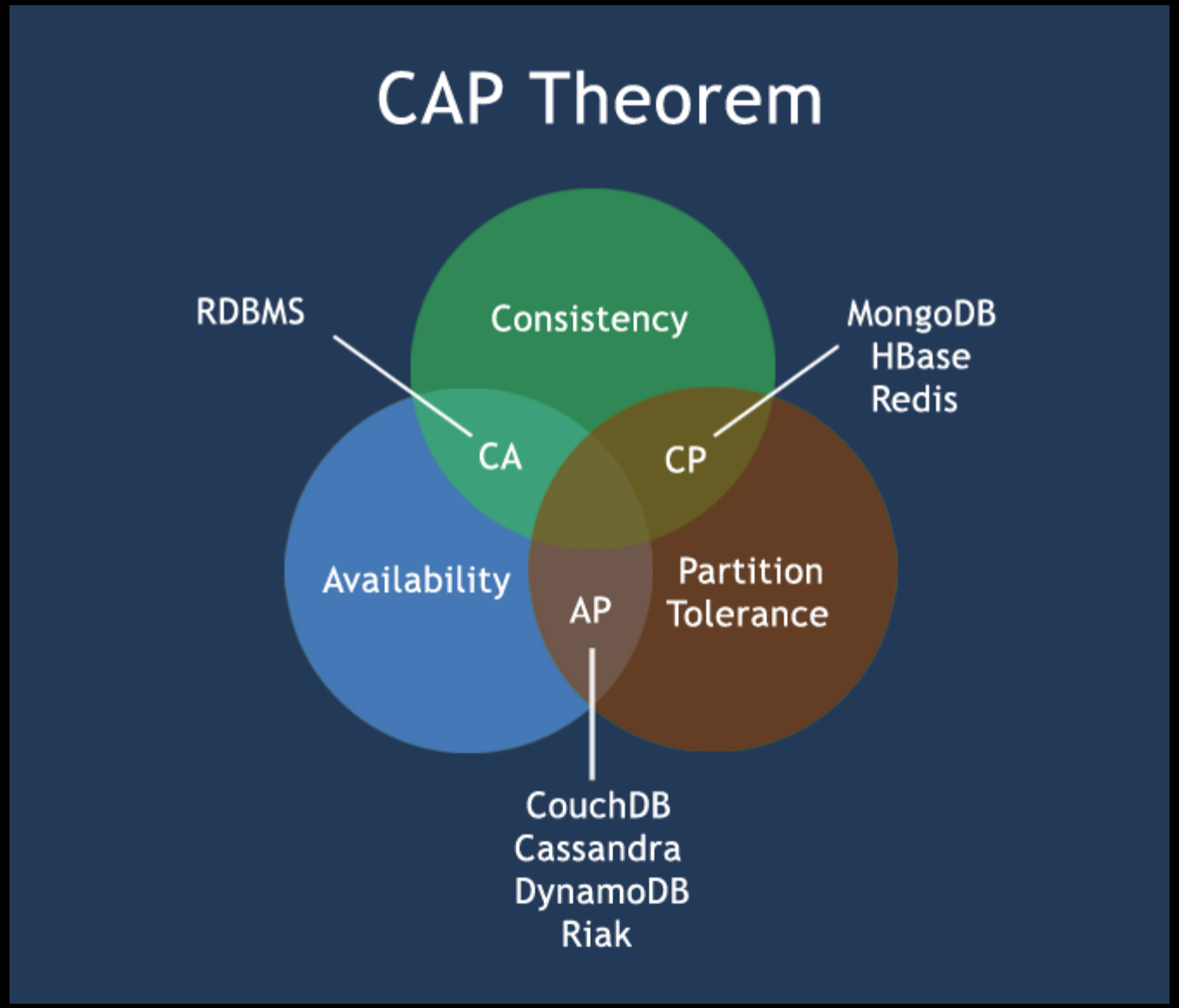
- Poprvé zmíněn v roce 1998 (Carlo Strozzi)
- Použil jej pro svojí vlastní databázi co neměla SQL interface
- V roce 2009 Eric Evans použil tento termín pro označení databází co jsou nerelační, distribuované a nemají vlastnosti atomicity, konzistence, izolace a trvanlivosti (ACID), tedy základní vlastnosti relačních databází
- Ve stejném roce byla první konference np:sql

Proč NoSQL v poslední době?

- Za posledních 20 let se enormně zvětšila kapacita úložných prostorů, ale rychlost dotazů a přenosů s tím nešla ruku v ruce.
- Twitter atd. ukládají tolik dat, že je není možné uložit na jeden disk, takže musí být klastrovaná
- Např. Twitter generuje více než 12 TB za den, tedy okolo 4 PB za rok.
- To samé platí pro další jako je Google, Facebook atd.
- Není to ale jen otázka webový aplikací, např. Large Scale Hadron Collider v Ženevě generuje okolo 12 PB za rok.

CAP teorém

- Je to základní teorém pro NoSQL resp. pro každý distribuovaný systém
- CAP teorém říká, že jsou tři základní požadavky, které je nutné brát v potaz když se navrhuje distribuovaná aplikace
- **Konzistence (Consistency)** - Data musí zůstat konzistentní i po provedení operace. Např. po uprav operaci všichni klienti uvidí stejná data.
- **Dostupnost (Availability)** - Systém musí být stále dostupný, tzn. že každý dotaz dostane odpověď, zda byl proveden úspěšně či selhal.
- **Partition Tolerance** - Systém musí být schopen stále komunikovat, i když je komunikace mezi severy nespolehlivá (např. pokud jsou severy replikovány na více uzlů a ty spolu nemohou řádně komunikovat)
- Teoreticky je nemožné splnit všechny tři najednou
- CAP říká, že základní požadavky na distribuované systémy musí splňovat dva ze tří požadavků
- Různé NoSQL databáze tedy podporují různé kombinace C, A, P
  - **CA -** data jsou konzistentní mezi všemi uzly, pokud jsou tedy všechny uzly online. Můžeme číst/zapisovat na a z kteréhokoli uzlu a být si zároveň jisti, že dat jsou stejná. Pokud ale vytvoříme partition mezi uzly, pak data budou nesychronizovaná i když bude partition znovu dostupná (nedojde k synchroznizaci).
  - **CP** - data jsou konzistentní mezi všemi uzly a zachovávají partition tolerance (zamezují desychroznizaci) tím, že pokud je uzel offline data jsou nedostupná.
  - **AP -** Systém je dostupný díky replikaci (různým klasterům) K synchronizaci dojde hned jak bude partition znovu dostupná. Nicméně není garantováno, že všechny uzly budou mít stejná data.



http://www.w3resource.com/mongodb/nosql.php

Výhody/nevýhody NoSQL

- **Výhody**
  - Vysoká škálovatelnost
  - Distribuce
  - Nižší cena
  - Flexibilní schéma
  - Částečně strukturovaná data
  - Nekomplikované vztahy
- **Nevýhody**
  - Neexistuje standardizace
  - Schopnosti dotazování je omezené
  - Eventuální konzistence

BASE

- CAP říká, že není možné plnit všechny tři požadavky najednou
- BASE systémy neuvažují konzistenci
- **B**asically **A**vailable říká, že systém negarantuje dostupnost (availability) z CAP
- **S**oft state říká, že systém se může měnit za běhu dokonce i bez vstupu. Díky eventual consistency modelu
- **E**ventual consistency říká, že systém bude konzistentní v určitém čase pokud během tohoto času nepřijde žádný vstup

ACID vs BASE

ACID	BASE
Atomicity	Basically Available
Consistency	Soft state
Isolation	Eventual consistency
Durable	

Kategorie NoSQL databází

Kategorie	**Popis**	**Databáze**
Dokumentově orientované (Document Oriented)	Data jsou uložena ve formě dokumentů. Například FirstName="Arun", Address="St. Xavier's Road", Spouse=[{Name:"Kiran"}], Children=[{Name:"Rihit", Age:8}]	CouchDB, Jackrabbit, MongoDB, OrientDB, SimpleDB, Terrastore atd..
XML databáze (XML database)	Data uložena ve formátu XML	BaseX, eXist, MarkLogic Server atd.
Grafové databáze (Graph databases)	Data uložena jako kolekce uzlů, kde každý uzel je analogií k objektu v objektovém modelu. Uzly jsou spojeny pomocí hran.	AllegroGraph, DEX, Neo4j, FlockDB, Sones GraphDB atd.
Klíč-hodnota databáze (Key-value store)	V případě klíč-hodnota uživatel ukládá data ve formátu bez schématu . Klíč může být řetězec, hash otisk, seznam, množina, setříděná množina a k němu je uložena hodnota	Cassandra, Riak, Redis, memcached, BigTable atd.

MongoDB

- *MongoDB je* Open Source databáze napsaná v C++.
- Pokud dojde ke zvýšení zátěže (např. více požadavků) pak databáze odpoví přidáním dalších uzlů.
- Využitelná pro uložení dat pro vysoko výkonnostní aplikace.
- MongoDB ukládá data jako dokumenty.
- Jedná se tedy o *dokumentově orientovanou databázi*.

```
FirstName="Arun",
Address="St. Xavier's Road",
Spouse=[{Name:"Kiran"}],
Children=[
    {Name:"Rihit", Age:8}
].
FirstName="Sameer",
Address="8 Gandhi Road".
```

- Jedná se o dva rozdílné dokumenty oddělené „.“

Základní vlastnosti MongoDB

- Dokumentově orientovaná databáze
- Snadno se programuje (C, C# and .NET, C++, Erlang, Haskell, Java, Javascript, Perl, PHP, Python, Ruby, Scala (via Casbah)),
- Snadná instalace
- Umožňuje indexovat kterýkoli atribut (např. FirstName=„Karel",Address=„Školská 87").
- Umožňuje škálovatelnost pomocí zrcadlení.
- Podporuje:
  - Update operaci nad celým dokumentem či nad určitými atributy
  - Map/Reduce dávkové zpracování a agregační funkce
  - JavaScript na straně serveru, díky tomu lze použít jeden programovací jazyk pro klienta i serverovou část
- Příklad Map/Reduce:
  - Map : master uzel přijme vstup, rozdělí jej na menší sekce a přepošle je do přiřazených uzlů.
  - Tyto uzly může provést také rozdělení a přeposlání nebo operaci vykonají a pošlou zpět odpověď/výsledek Master Node.
  - Reduce : master uzel provede agregaci částečných výsledků a vytvoří finální výstup

Databáze, dokumenty a kolekce v MongoDB

- Databáze
  - Podpora více databází na jednom serveru
  - Je možné vytvářet databáze za běhu – není třeba vytvořit db než s ní začneme pracovat.

Dokumenty

- *document* je základní jednotkou pro uložení dat.
- *document* používá se JSON (JavaScript Object Notation) pro uložení dat.
- Příklad:
  - { site : "w3resource.com" }
- Termín "objekt" pak je synonymum pro dokument.
- Dokumenty jsou analogií k tabulkám v relačním modelu
- Insert, update a delete operace mohou být prováděny nad kolekcí (collection).

SŘDB	**MongoDB**
Tabulka	Kolekce (Collection)
Sloupec	Klíč (Key)
Hodnota	Hodnota (Value)
Záznamy / řádky	Dokument/objekt (Document / Object)

Základní datové typy

Datový typ	Popis
string	Prázdny řetězec nebo kombinace znaků.
integer	čísla.
boolean	True nebo False.
double	Číslo s desetinou částí
null	Jako v relačním modelu, ani nula ani prázdné, neznámá nebo neaplikovatelná hodnota
array	Seznam hodnot
object	Entita, kterou můžeme využít k programování (hodnota, proměnná, funkce nebo datová struktura)
timestamp	64 bit hodnota času unikátní v rámci jedné instance MongoDB. Prvních 32 bitů ukládá počet sekund 1 ledna 1970 Druhá polovina je ordinální číslo pro práci s danou sekundou
Unikód	UTF-8
ID objektů	Každý objekt/dokument musí mít svoje unikátní ID Uloženo jako 12 bitová hodnota BSON(Binary JavaScript Object Notation) ==binární representace JSON) Složeno ze 4-byte timestamp, 3-byte id počítače, 2-byte id procesu a 3-byte čítač.

Kolekce

- Kolekce může ukládat několik dokumentů a odpovídá tabulce v relačním modelu
- Kolekce může ukládat dokumenty různých typů (MongoDB nemá definované schéma)
- Příklad dvou dokumentů patřících do stejné kolekce, ale mající jinou strukturu:
  - {"tutorial" : "NoSQL"} {"topic_id" : 7}
- Kolekce je vytvořena při vložení prvního záznamu-dokumentu
- tzv. **Capped collections** jsou kolekce, kde jsou data uložena ve stejném pořadí a jakém byla uložena

Práce s daty – vložení dokumentu

1. Vytvoření dokumentu

1.
```
document=({
"user_id" : "ABCDBWN",
"password" :"ABCDBWN" ,
"date_of_join" : "15/10/2010" ,
"education" :"B.C.A." ,
"profession" : "DEVELOPER",
"interest" : "MUSIC",
"community_name" :[
   "MODERN MUSIC","CLASSICAL MUSIC","WESTERN MUSIC"],
"community_moder_id" :[
   "MR. BBB","MR. JJJ","MR MMM"],
"community_members" : [
   500,200,1500],
"friends_id" : [
   "MMM123","NNN123","OOO123"],
"ban_friends_id" :[
   "BAN123","BAN456","BAN789"]
});
```

2. Vložení dokumentu `db.userdetails.insert(document)`
3. Zobrazení dat `db.userdetails.find()`

Práce s daty – aktualizace dokumentu

- Metoda `update (criteria, objectnew, upsert, multi)`
  - criteria – dotaz, který vrátí dokument co chceme aktualizovat;
  - objectnew- specifikace co chceme v dokumentu aktualizovat nebo můžeme využít operátor $ (např. $inc...).
  - upsert – pokud je true, pak se vytvoří nový dokument pokud žádný neodpovídal kritériím.
  - multi – pokud true, pak jsou v případě, že kritériím odpovídá více dokumentů aktualizovány všechny nebo jen jeden (to je výchozí akce).

Práce s daty – update

- Dokument {"_id":11, "item" : "Divine Comedy", "stock":2}

- db.books.update({ item: "Divine Comedy" }, { **$set**: { price: 18 }, **$inc**: { stock: 5 } })

- Výsledek:
- { "_id" : 11, "item" : "Divine Comedy", "price" : 18, "stock" : 7 }

Práce s daty – update, přidání atributu

- Přidání nového atributu k dokumentu, provede se pokud aktualizovaný atribut v dokumentu není

```
db.bios.update(
    { _id: 3 },
    { $set: {
            mbranch: "Navy",
            "name.aka": "Amazing Grace"
        }
    }
)
```

Práce s daty – update, odebrání atributu

- `db.bios.update( { _id: 3 }, { $unset: { birth: 1 } } )`

Práce s daty – update, odebrání dokumentu

- `db.userdetails.remove({ "user_id":"testuser" })`
- Odebrat všechny
- `db.userdetails.remove({})`

Práce s daty – find

- `db.userdetails.find({"education":"M.C.A."}).pretty()`
- `SELECT * FROM userdetails WHERE education="M.C.A.";`
- Projekce
- `db.userdetails.find({"education":"M.C.A."},{"user_id" : 1}).pretty();`
- `SELECT user_id FROM userdetails WHERE education="M.C.A.";`
- Projekce více atributů
- `db.userdetails.find({"education":"M.C.A."},{"user_id" : 1,"password":1,"date_of_join":1}).pretty();`
- `SELECT user_id,password,date_of_join FROM userdetails WHERE education="M.C.A.";`
- Všechny atributy vyjma user_id
- `db.userdetails.find({"education":"M.C.A."},{"user_id" : 0}).pretty();`

Práce s daty – třídění

- `collection.find().sort( {column1:1or -1 [, column2:1 or -1] });`
- 1 .. Ascending
- -1 .. descending

- `db.userdetails.find().sort({"education":1}`
- `SELECT * FROM userdetails ORDER BY education`

- `db.userdetails.find().sort({"education":1,"password":-1})`
- `SELECT * FROM userdetails ORDER BY education, password DESC;`

Práce s daty – limit

- `db.userdetails.find().limit(2).pretty();`
- `SELECT * FROM userdetails LIMIT 2;`
- Přeskočit první dva záznamy a pak vrátit vše
- `db.userdetails.find().skip(2).pretty();`

$in,$nin a $or,$nor

- $in zkontroluje zda daný parametr je v definovaném rozmezí argumentů.
- $or operátor ověří, zda je některá z definovaných podmínek splněna
- `db.userinfo.find({"age":{$in:[19,22]}}).pretty();`
- `SELECT * FROM userinfo WHERE age IN(19,22);`

- `db.userinfo.find({ "sex" : "Male" , $or : [ { "age" : 17 } , { "date_of_join" : "17/10/2009" } ] } ).pretty();`
- `SELECT * FROM userinfo WHERE sex="Male" AND (age=17 or date_of_join="17/10/2009");`

$and, $not

- db.student.find({$and:[{"sex":"Male"},{"grd_point":{ $gte: 31 }},{"class":"VI"}]}).pretty();
- SELECT * FROM student WHERE sex="Male" AND grd_point>=31 AND class="VI";
- db.student.find({"age": { $not: {$lt : 12}}}).pretty();
- SELECT * FROM student WHERE age>=12;

Null hodnoty ($type, $exists)

- $type operátor přiřazuje hodnoty na základě jejich BSON typu
- $exist operátor kontroluje existenci daného atributu
- `db.testtable.find( { "interest" : null } ).pretty();`
- Všechny dokumenty co nemají atribut interest
- `db.testtable.find( { "interest" : { $exists : false } } ).pretty();`

Tečková notace

- Pomocí tečky se dostáváme k atributům, dokumentům uvnitř jiných dokumentů
- `db.testtable.find({"extra.community_name" : "MODERN MUSIC"}).pretty();`

- `db.testtable.find({"extra.community_name" : "MODERN MUSIC","extra.friends.valued_friends_id":"harry"}).pretty();`

Operátory porovnání

- (>) greater than - $gt
- (<) less than - $lt
- (>=) greater than equal to - $gte
- (<=) less than equal to - $lte
- (<>, !=) not equal to - $ne

- `db.testtable.find({age : {$gt : 22}}).pretty();`
- `SELECT * FROM testtable WHERE age >22;`