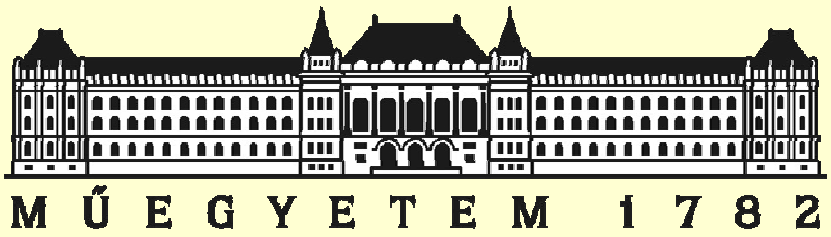
NoSQL @ Adatb haladóknak

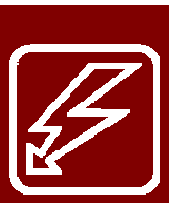
NoSQL adatbázisok

Trencséni Márton
mtrencseni@gmail.com



Adatbázisok haladóknak 2012.

2012. szeptember 25.



IMDB @ Adatb haladóknak

Kérdés

- Használtatok-e már NoSQL terméket?
- Mikor használnátok NoSQL terméket?

IMDB @ Adatb haladóknak

Az előadóról

- 1999-2004: BME, Info
- 2002-2008: ELTE, Fizikus

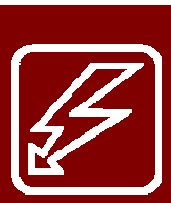
- 2004-től: versenyszféra, pl. Graphisoft
- 2009-2012: Saját NoSQL cég, Scalien

IMDB @ Adatb haladóknak

Előadás terve

- SQL és NoSQL története
- Konkrét példa: MongoDB
- Informatikai környezet, SaaS
- Elosztott aspektusok
  - Skálázhatóság
  - Replikáció
  - Konzisztencia, CAP
- Konklúzió

Kérdések, válaszok

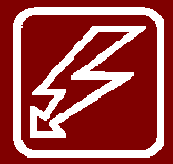


SQL történet

- 196x, 197x: COBOL, CODASYL
- 1970, IBM: Edgar F. Codd – *A Relational Model of Data for Large Shared Data Banks*
- 197x, IBM: Chamberlin, Boyce – *SEQUEL*
- 197x, IBM: System R
- 1974, U.C.Berkeley: Ingres
- 197x, Larry Ellison: Relational Software, Inc. (Oracle)
- 1986: ANSI SQL (1st)
- 197x – 199x: Aranykor

SQL ma

- Oracle
- Microsoft SQL Server
- IBM DB2, Informix
- Teradata
- Ingres
- Mysql (Oracle)
- Postgresql



Dominancia

- Relációs modellben gondolkozunk
  (Gyakran NoSQL esetben is)
- Legyőzte a hálós modellt a 70-es években
- Legyőzte az objektum modellt a 90-es években
- SQL adatbázis tovább él, mint az elérésre használt programozási nyelv
- SQL adatbázis tovább él, mint a host operációs rendszer

Reneszánsz

- 2005 előtt: RDBMS/SQL legtöbb website mögött
- 2005: Google cikkek
  - Ghemawat, Gobioff, Leung: *Google File System*
  - Chang, Dean, Ghemawat, et al.: *Bigtable*
  - Mike Burrows: *Chubby*
  - Chandra, Griesemer, Redstone: *Paxos made live*
  - Dean, Ghemawat: *MapReduce*
- DeCandia et al., Amazon: *Dynamo*
- Facebook: Cassandra

Google cikkek

- Mit újitott a Google és az Amazon?
- Nincs új algoritmus vagy adatszerkezet
- De: Létező ötletek újfajta alkalmazása
- Rendszertervezésről szóló cikkek
- Új, **elosztott architektúra**
- **Nem relációs modell, nincs SQL**
  - **Mert nehéz elosztott rendszerben SQL-t csinálni (bonyolult lekérdezések, tranzakciók, ACID garantálása)**

IMDB @ Adatb haladóknak

NoSQL “forradalom”

- Google, Amazon technológiák: nem open-source
- 2005-2010: Elkezdünk open-source NoSQL technológiákat fejleszteni a Google/Amazon ötletek alapján
- Maximum: 2009, új projektek tűnnek fel és el hetente
- 2012 és tovább: ki marad életben?
  - MongoDB (új Mysql)
  - Hadoop (Big Data analitika)
  - Cassandra (EC, CF, Georeplikáció)

IMDB @ Adatb haladóknak

NoSQL ma

Company	Product	Customers	Funding
10gen	**MongoDB**	**100-1000**	**$ 73,400,000**
Cloudera	**Hadoop**	**10-100**	**$ 76,000,000**
Couchbase	CouchDB	10-100	$ 31,000,000
Basho	Riak	10-100	$ 26,000,000
Hortonworks	Hadoop	10-100	$ 20,000,000
DataStax	**Cassandra**	**~100**	**$ 13,700,000**
Neo	Neo4J	10-100	$ 13,100,000
Citrusleaf	Citrusleaf	1-10	$ 2,000,000

Oracle market cap: $160 Milliárd…

MongoDB

- C++
- Mysql helyére akar lépni, Chevy modell
- Light-weight use-case
- Érdekes termékfejlesztési filozófia:
  - Advanced dolgod eleinte nem működtek
  - Javítgattak rajtuk, átírták
  - Pl. replikáció
  - Gyorsan tudtak növekedni, mert a legtöbb usernek ezek nem kellettek

MongoDB

- Nagyon könnyű elindulni
- Letöltöd és elindítod az .exe-t
- Rengeteg kliens library
- Rengeteg port (pl. Django on MongoDB)
- Nagy netes lefedettség (Google, Stackoverflow)
- Gyors és állandó releasek, jelenleg 2.2.x

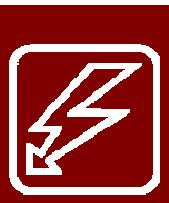
MongoDB

- Nincs séma
- Document store = JSON store
- JSON dokumentum példa:

```
{
   "company":"Acme",
   "employees":
   [
      {"firstName":"John", "lastName":"Doe" },
      {"firstName":"Anna", "lastName":"Smith"},
      {"firstName":"Peter"}
   ]
}
```

MongoDB

- Funkcionális API
- ```
  for (var i = 1; i <= 20; i++)
    db.things.save({“x”:4, “j”:”alma”+i});
  for (var i = 1; i <= 20; i++)
    db.things.save({“x”:4, ”alma”+i:”j”});
  db.things.find({“x”:4});
  db.things.find().limit(3);
  ```
- Indexek
- Tranzakciók nincsenek
- Atomi műveletek dokumentumokon
- Server side Javascript SPs (lassú)
  ```



MongoDB

- Single server architecture
- Aszinkron replikáció replica set-eken belül
- W paraméter
- Auto-sharding, pre-sharding

MongoDB erősségek

- Gyorsan el lehet indulni
- Sok információ az Interneten
- Jól működik kevés, közepesen sok adattal
- … ha nem használod nagyon az elosztott dolgokat
- Nagyon jól hangolható
- Bottom-up marketing model
- Mysql már az Oracle tulajdona…
- Trendy

NoSQL informatikai környezete

- Miért van létjogosultsága egy ilyen típusú terméknek?

- Web
- Software-as-a-service

Webes adatok

- Tipikus SQL: CRM, ERP

- Kevésbé struktúrált
- Változó struktúra
- Piszkosabb
- Kevesebb logikai kapcsolat (normalizáció)
- Szöveges
- Keresés szöveg alapú
- Stb.

IMDB @ Adatb haladóknak

SaaS előnyei

- Bárhonnan elérhető
- Cross platform
- Rapid development: állandó bugfixek, featurek
- Havi bevételek vs. verziónkénti bevételek
- A/B tesztelés
- Rengeteg analitika
  - Funkcionális: mit nyomogat a user
  - Adat: mit hoz létre a user
  - Web: honnan jön a user, stb.

IMDB @ Adatb haladóknak

Saas kihívásai: rapid development

- Agile, Scrum, Sprints, stb.
- Pl. sikeres magyar startupoknál 2 hetes **iterációkban** dolgoznak
- Minél hamarabb rakd ki produkcióba, aztán meglátjuk és iterálunk
- Gyorsan változó kód
- **Gyorsan változó adatséma**
- Nem akarnak SQL sémákat létrehozni és azoknak az életciklusát követni:
  - Fejlesztőgépeken
  - Tesztkörnyezetben
  - Produkciós környezetben
  - Continuous Integration (CI), pl. Jenkins
- **Legyen dinamikus JSON,** esetleg Protocol Buffers

Egyéb szempontok SQL ellen

- Framework-öt használnak
  (RoR, Django, stb.), amik általában eleve egy ORM rétegen keresztül mennek, eleve “mindegy”
- Always-on: Adatbázis feltétel miatt ne legyen hiba, majd utólag kijavítjuk!
- Drága? Know-how?
- Egyenlőre azért ne temessük az SQL-t, sok webes cégnél a fontos adatokat továbbra is SQL-ben tárolják, de aktívan próbálnak áttérni NoSQL megoldásokra

IMDB @ Adatb haladóknak

Mit veszítünk?

- Nincs SQL ☺
- Pl. ha észreveszed, hogy hibás vagy piszkos adat van, akkor ennek helyrehozása már nem olyan egyszerű, mint “UPDATE … WHERE …”
- Márpedig az eddigiek miatt sok ilyen lesz ☺
- Egyszerű analitikát sem lehet csinálni
- Tranzakciók ☹
  - A fejlesztési ciklus vége fele kezd el fájni, de akkor nagyon
- Nincs séma, “Mi van az adatbázisban?”
  - Pl. jön egy új alkalmazott
- Hozzáférési jogosultságok
- Stb.

SaaS kihívásai: adat tárolása

- Nagy mennyiségű
  - adat és
  - forgalom
- az üzemeltetőnél

- Adat ne vesszen el
- Szolgáltatás álljon rendelkezésre és legyen gyors

SaaS architektúra

- Adatbázis (Mysql, Pgsql, MSSQL, Mongo…)
- Cache szerver (Memcache, Redis, Mongo)
- Alkalmazás (PHP, Python, Java, .NET, Rails)
- Web szerver (Apache, IIS)
- Browser (Firefox)
  - Queue szerver
  - Mail szerver
  - Storage szerver
  - Virtualizáció
  - Külső valami-as-a-Service szolgáltatók
  - Facebook, Twitter, stb.

Amazon Web Services

- Sok cég, főleg startup/kis cég használja, Netflix
- Az AWS kínál database-as-a-Service-t is HTTP-n keresztül, ez NoSQL, de tapasztalatom szerint a legtöbb ügyfél nem használja ezt, hanem az EC2 gépein saját adatbázist futtat
  - Database-as-a-service még nem futott fel, ha egyáltalán fel fog
  - Miért? “Az adat nagy érték, nálam legyen.”

IMDB @ Adatb haladóknak

Valójában mennyire sok adatról beszélünk?

- A NoSQL világban a Google, Facebook, Amazon a standard referencia
- Valójában rajtuk és pár más cégén kívűl (pl. LinkedIn) nincs olyan sok ONLINE adat egy-egy cégnél, főleg Magyarországon
- Egy hisztogramon az adatbázis mérete valószínűleg exponenciálisan leesik?
- De: analitikához valóban sok adatot lehet gyűjteni, ~10-100G/nap Magyarországon is (Hadoop)

Nagy mennyiségű adat és forgalom

- NoSQL use-caseket külön kell választani:
  - Primary database
    MongoDB, Cassandra, Riak
    (Online Request Processing vs. OLTP)
  - Cache database
    MongoDB, Redis
  - Analytics (Big Data)
    Hadoop

IMDB @ Adatb haladóknak

NoSQL mint elosztott adatbázis

- Lényegesen új elméleti eredmény nincsen
  pl. új algoritmus vagy adat struktúra
- Viszont: új műszaki megoldások kipróbálása

- Eddig: Csináljunk egy SQL adatbázist, és utána nézzük meg hogyan lehet elosztani

- Google, Amazon: Csináljunk egy elosztott adatbázist, és utána nézzük meg milyen sémát/lekérdezés tudunk rárakni

- Ellenpélda: MongoDB

Lehetőségek

- Vertical scaling:
  több memóriát, diszket veszünk
- **Horizontal scaling: (NoSQL)**
  újabb gépeket rakunk be

- Shared-something:
  Pl. osztott diszk egy SAN-on keresztül,
  pl. Oracle RAC
- **Shared-nothing (Stonebraker): (NoSQL)**
  A clusterben lévő gépek nem osztanak meg CPU/diszk/memóriát, csak hálózaton beszélnek
- L. Ellison: “Shared-nothing is good for nothing.”

Skálázhatóság

- Lineáris skálázhatóság

- Storage scalability
- Read scalability (throughput)
- Write scalability (throughput)

2x annyi szerver, 2x “gyorsabb”?

- Write Latency: nem csökken
- Read Latency: csökkenhet, ha új szervereken befér a RAM-ba
- Sok műveletből álló nem párhuzamosítható (tipikus) utasítás csomag végrehajtási ideje = SUM(latency)
  Olvasásokból adódóan csökkenhet, ha írások dominálják nem csökken

2x annyi szerver, 2x áteresztőképesseg?

- 1 darab szerver
- 1 darab kliens gépen 1 szál
- Addig növeljük a szálak számát, amíg több parancsot nem tud kiszolgálni a szerver, azaz szaturáljuk a szervert
- Berakunk még egy szervert, ki tud-e szolgálni több parancsot? Igen, ha tovább növeljük a szálak számát

2x annyi szerver, 2x áteresztőképesseg?

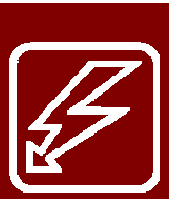
- Tehát 2 szerver, kétszer annyi kliens szál
- Kérdés: 2x lesz-e a throughput, azaz lineáris-e?
- Válasz: ha implementációs és hardveres korlátokat még nem érünk el, akkor lehetne…
- Ütközések: ha a két kliens véletlenszerűen dönti el, hogy hova küldi a következő parancsot, akkor ½ eséllyel ugyanoda küldik, és egymásra fognak várni!

2x annyi szerver, 2x áteresztőképesség?

- Ütközések miatt szub-lineáris lesz a skálázhatóság

				Clients			
		1	2	3	4	5	6
	1	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.49	1.66	1.73	1.79	1.82
	3	1.00	1.67	2.06	2.28	2.40	2.50
Servers	4	1.00	1.74	2.27	2.60	2.86	3.04
	5	1.00	1.80	2.40	2.85	3.20	3.44
	6	1.00	1.83	2.50	3.05	3.44	3.78

** Házi feladat: Mi a képlet?*



2x annyi szerver, 2x áteresztőképesseg?

- Write: Általában ide is, oda is ír, pl. Consistent Hashing-et használó rendszerek
- Read: Ha több replikán megvan az adat, akkor véletlenszerűen választja ki a kliens, honnan olvasson (ScalienDB)
  - Lehetne koordinálni, hogy elkerüljék az ütközést, de ilyen megoldásról nem tudok

Skálázhatóság

- Mikor van triviálisan lineáris skálázhatóság?
- Ha nem kell egymásra várakozni, nincs nagy kommunikációs overhead = analitika

- Pl. ha egy batch jobot futtatunk Hadoop-on.
- Szétküldi a jobot, ~1sec
- A gépek futtatják, 10min
- Összefésülés, tfh. gyors, ~1sec
- 2x annyi gép, 10min helyett 5min

- “Hadoop is scalable”

Replikáció

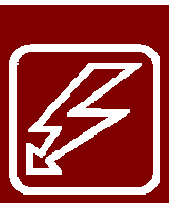
- Alap kérdés egy elosztott rendszerben: Milyen algoritmus vagy protokoll szerint replikáljunk?
- NoSQL világban elterjedt az **Eventual Consistency**
- Másik véglet az Immediate Consistency vagy **Strong Consistency** vagy Consistent Replication

Consistency

- ACID =
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- Tranzakciókra vonatkozik, nem replikációra!
- Replikációnál a Consistency mást jelent!

Consistency

- Általánosan elfogadott definíció nincs, mást jelent pl. Cassandra-nál, Hbase-nél és MongoDB-nél, ráadásul egy-egy adatbázisnak több üzemmódja van
- Alapvetően az a kérdés, hogy ha beleírsz egy értéket az adatbázisba, majd később visszakérdezed, akkor mit fogsz visszakapni?
  - Arra gondolj, hogy az a szerver, aki a READ-et kiszolgálja, nem biztos, hogy ugyanaz, mint aki a WRITE-ot átvette előtte
- (A gyakorlatban a Durability kérdésével is összefügghet, ld. MongoDB példa később)



Eventual Consistency

- Írás után van egy időablak, amíg ha egy másik szerver szolgálja ki az olvasást, akkor lehet, hogy régi adatot lát, de idővel *(eventually)* mindegyik replikához el fog érni az írás művelet
- Másik: írás után nem biztos, hogy minden replikán ugyanaz lesz az adat (konzisztens), de “idővel igen”
- Valójában nem adható időkorlát, mert van amikor a kliensnek kell feloldania a konfliktust

- Problémák:
  - Elfogadható-e, hogy egy ideig régi adatot kapok vissza? Ha nem, akkor ütközik a read scalability-vel.

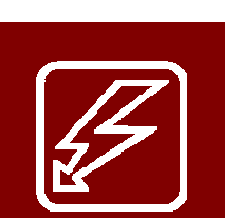
MongoDB példa (WriteConcern)

- Sok írási módot meg lehet adni:
  - A kliens library átveszi, de el se küldi a szervernek
  - Elküldi a masternek
  - Elküldi a masternek, journal
  - Elküldi a masternek, journal, fsync
  - W, majority: Ennyi darab replikára kikerült
- Read:
  - Primary
  - primaryPreferred
  - Secondary
  - secondaryPreferred
  - Nearest

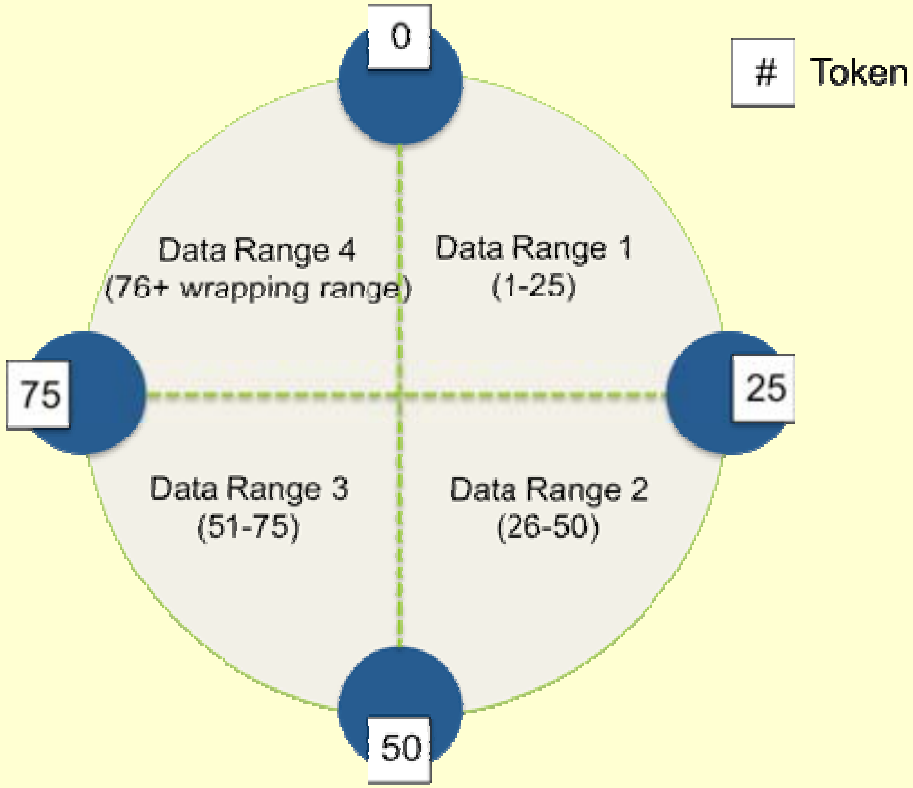
Cassandra példa

- Alapvető architektúra replikáció és skálázhatóság szempontjából:
  Consistent Hashing

- Adott kulcshoz kiszámol egy hash értéket
- Az egy pozíció egy körben
- A kör fel van osztva szerverek között
- A pozíció utáni N=3 szerverre kerül az adat



Cassandra példa



Cassandra példa

- N = replikációs faktor (3)
- W = hány replikára küldje az írást (2)
- R = hány replikáról olvasson (2)
- Verzió timestampekkel tárolja az adatot
- Írás: Node átveszi az írást, generál egy új verzióvektort, kiírja diszkre, majd továbbkuldi még W-1 másiknak
- Olvasás: R helyről olvas, ha nincs ütközés visszaad egy értéket, ha van divergencia akkor többet ad vissza (nem atomi)

Paxos

- Konszenzus algoritmus
- Ez (matematikailag) bizonyítottan minden hálózati probléma esetén konzisztens
  - Csomagok sorrendje felcserélődik
  - Csomagok elvesznek
  - Hálózati partíció
  - Szerver kiesik, visszajön
- Nem kell verziózni, nem lehet szétágazó verzió

Paxos

- Replikákat fogjuk fel egy állapotgépnek
  - Állapot = Adatbázis
  - Állapotátmenetet = Adatbázis parancs
  - Parancsok egymásután = Replikált log
- Paxos protokollal az állapot átmeneteket replikáljuk, és garantálja, hogy mindegyik replika ugyanazt a replikált logot látja
- Többségi algoritmus, így előfordulhat, hogy egy replika replikált logja le van maradva

Paxos

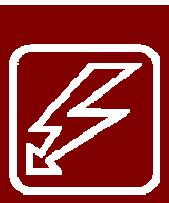
- Hasonló mint az elosztott Commit protokollok
- 3 fázis:
  - Prepare
  - Propose
  - Learn
- 3 szerep:
  - Proposer
  - Acceptor: perzisztens memória egység
  - Learner

Egyszerű többségi szavazás nem jó

- Proposer szétküldi a következő adatbázis parancsot
- Acceptorok befogadják
- Deadlock-hoz vezethet
- Ezért kell valamiféle Undo vagy Overwrite mechanizmus, de hogy garantálod, hogy ha a többség elfogadta, akkor nem lesz Undo/Overwrite-olva?
- Erre ad optimális megoldást a Paxos
- Lásd Leslie Lamport: *Paxos made simple*
  Alapvető számításelméleti eredmény, érdemes ismerni

Paxos algoritmus

- **Prepare fázis:**
- Proposer: be akarja replikálni a V adatbázis parancsot, küld egy PREPARE üzenetet >N/2 Acceptornak, lokálisan növekedő ProposalID-val, V-t nem tartalmazza.
- Acceptor: Ha kap egy PREPARE-t ProposalID-val, és nincs nagyobb ígérete, akkor küld egy PROMISE üzenetet a Proposer-nek, és onnantól <= ProposalID-ra REJECT-et küld (ki kell írnia diszkre ProposalID-t); ha elfogadott előtte másik V' parancsot, akkor a PROMISE válaszba belerakja a V' parancsot és azt a ProposalID-t is.
- **Propose fázis:**
- Proposer: Ha kap >N/2 PROMISE üzenetet, akkor átléphet a Propose fázisba. Ha az összes Acceptor üres PROMISE-t küldött, akkor a saját V-t használja a továbbiakban, egyébként a legnagyobb visszakapott ProposalID-jű V:=V' parancsot. Most küld egy PROPOSE üzenetet a >=N/2 Acceptornak, tartalmazza V parancsot és a saját eredeti ProposalID-t.
- Acceptor: Ha nincs nagyobb igérete, akkor elfogadja a PROPOSE üzenetet és megjegyzi a ProposalID-t és V-t (kiírja diszkre), majd küld egy ACCEPTED üzenetet.
- Proposer: Ha >N/2 ACCEPT üzenetet kapott, akkor most már biztos, hogy V lett a választott érték, ha később egy másik Proposer is bejön az is erre fog jutni. Ezért az összes Learner-nek szérkürtöli LEARN parancsban, hogy konszenzus van, V az érték.
- **Lean fázis:**
- Learner: LEARN hatására végrehajtja a V adatbázis parancsot lokálisan (beírja a lokális adatbázisba).



Paxos

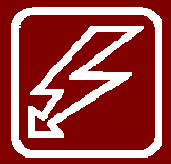
- Pl. saját termék (ScalienDB) Paxos alapú replikációt használt
- https://github.com/scalien/scaliendb
- Jelenleg egy másik Paxos-os termék van a NoSQL piacon, kis cég, kevés tőkével, sajnos nem open-source: **CitrusLeaf**
- Általában azt mondják, hogy a Paxos túl lassú, mert
  - 2+1 fázisból áll (1 csökkenthető)
  - diszkre kell írni (csak egyszer, és diszk írás nélkül nem lehet erős konzisztenciát csinálni)

CAP

- CAP háromszög
- C = Consistency
- A = Availability
- P = Partition [tolerance]

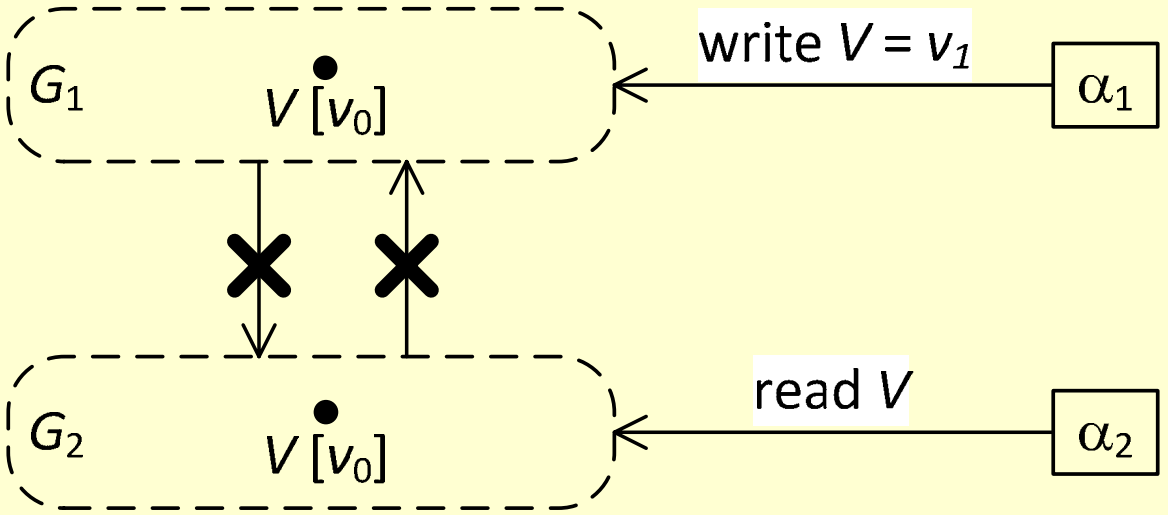
CAP

- “Egy elosztott rendszer akkor **konzisztens**, ha egy adategység értékét bármely csomóponttól lekérdezve ugyanazt az értéket kapjuk.”
- “Egy elosztott rendszer **rendelkezésre áll**, ha minden működő csomóponthoz érkező kérésre válaszol, tehát a csomópontokon futtatott algoritmusoknak véges idő alatt be kell fejeződniük.”
- “A hálózat **partíciója** úgy modellezhető, hogy a hálózat egyik csomópontjából a másikba küldött üzenetekből tetszőleges számú elveszhet. Egy rendszer tehát akkor partíció toleráns, ha a kérésre hálózati partíció esetén is helyes választ ad.”



CAP

- “A CAP-tétel röviden úgy fogalmazható meg, hogy elosztott rendszerben a hálózat partíciója esetén a rendszer műveletei nem lesznek atomiak és/vagy az adategységei elérhetetlenek lesznek.”



CAP kritika

- Formalizált verzió triviális?
- A és P összefügg: ha nincs partíció, azaz minden node tud kommunikálni, akkor az A mint kikötés értelmetlen, mert akármelyik node-hoz bejön egy kérés, az továbbíthatja egy másik, kijelöltnek (proxy)
- → CA kombináció értelmetlen
- AP kombináció: bármit visszaadhat ☺
- Definíció szerinti A-t senki sem várhatja el általánosságban; nem túl hasznos a rendszer ami ezt tudja, még Amazon szerint is, akinek volt ilyesmi
- Negatív eredmény; sokkal érdekesebb a Paxos algoritmus, ami egy pozitív eredmény
- “CAP – válassz kettőt”

Konklúzió

- Elméleti oldalról az elosztott adatbázisok az érdekesek (nem a NoSQL)
- Mérnöki oldalról a kettő metszete és kiterjesztése, pl. tranzakciók
- **Webes analitika**: azaz a Big Data világában a **Hadoop** megkerülhetetlen és ma már standard az USA-ban
- **OLRP:**
  - MongoDB, Redis elterjedtek
  - Cassandra
  - Mysql-t megvette az Oracle, így jövője bizonytalan
  - Pgsql

Érdekes témák

- Tranzakciók (nem megoldott, Mongoban *atomic*)
- Geo-replikáció (nem megoldott, Cassandrában a legjobb)
- Adatmodell (dinamikus legyen, de azért legyen séma)

Érdekes témák

- Új Google cikkek, adatbázisok:
  - Dremel
  - Pregel (large-scale graph processing)
  - **Spanner** (geo-replicated, transactional, synchronous replication)
- Elosztott algoritmusok (mérnöki tudomány és matematika találkozása)
  - Elosztott algoritmusok formalizálása, pl. TLA-ban
  - Új algoritmusok
  - Pl. quorum tagság változása, leader election, hibás vagy rosszindulatú node, skálázhatóság

Ajánlás

- Próbáld ki a MongoDB-t
- Ha érdekel a rendszertervezés téma, olvass Google cikkeket
- Nézzél bele a forráskódba, minden open-source:
  - MongoDB, C++
  - Hadoop, Java
  - Cassandra, C++
  - ScalienDB, C++
- Ha érdekelnek elosztott algoritmusok, sok kihívás, szép eredmények lehetősége

IMDB @ Adatb haladóknak

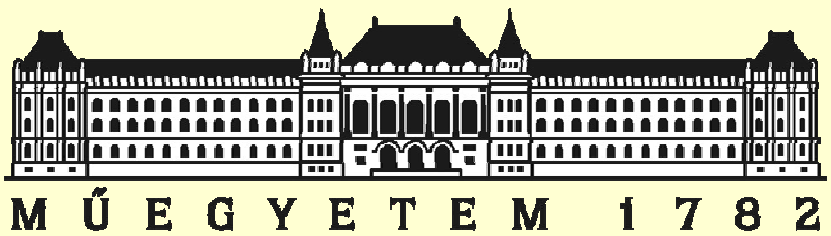
Kérdések…

- …válaszok (?)

IMDB @ Adatb haladóknak

Köszönöm a figyelmet!

Trencséni Márton
mtrencseni@gmail.com



Adatbázisok haladóknak 2012.

2012. szeptember 25.