A Facebook adattárháza

Trencséni Márton
mtrencseni@gmail.com
“info99”

Néhány szót rólam

1. BME-n végeztem 2004-ben műszaki informatikusként
2. Adatbázisok tárgy már akkor is volt :)

Utána:

- ELTE, fizikus
- Graphisoft és kisebb magyar cégek (Budapest, 2008-2012)
- Scalien (Budapest, 2008-2012)
  (saját NoSQL startup, https://github.com/scalien/scaliendb)
- Prezi (Budapest, 2013-2015)
- **Facebook (London, 2016-2017) - Data Engineer**
- Fetchr (Dubai, 2017-2018)

Tartalomjegyzék

Mi az a “Data Engineer”?

Responsibilities

- Manage data warehouse plans for a product or a group of products.
- Interface with engineers, product managers and product analysts to understand data needs.
- Build data expertise and own data quality for allocated areas of ownership.
- Design, build and launch new data models in production.
- Design, build and launch new data extraction, transformation and loading processes in production.
- Support existing processes running in production.
- Define and manage SLA for all data sets in allocated areas of ownership.
- Work with data infrastructure to triage infra issues and drive to resolution.

Minimum Qualifications

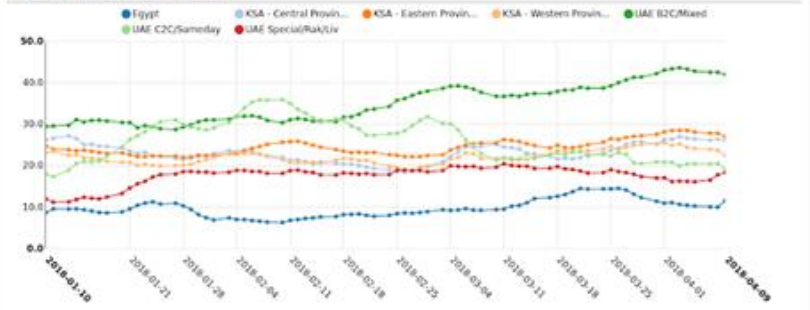
- 2+ years experience in the data warehouse space.
- 2+ years experience in custom ETL design, implementation and maintenance.
- 2+ years experience working with either a MapReduce or an MPP system.
- 2+ years experience with object-oriented programming languages.
- 2+ years experience with schema design and dimensional data modeling.
- 2+ years experience in writing SQL statements.
- Experience analyzing data to identify deliverables, gaps and inconsistencies.
- Experience managing and communicating data warehouse plans to internal clients.

Xinsights ☆

Topline Ops - Explanation

WARNING: The charts here show a 7 day rolling average to make it more readable. Friday is excluded. A **Fully Utilized Driver (FUD)** is a driver with more than 30 P+D dispatches that day. **FUD DPD** is the DPD of FUD drivers. — Questions/comments to Marton Trencseni. [Code]

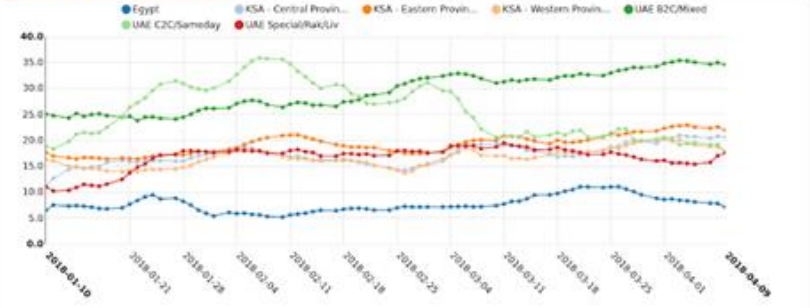
Topline Ops - P+D Dispatches Per Driver



Topline Ops - Latest Dispatches Per Driver

Region Fleet	_timestamp	P+D Dispatched	P+D Dispatch Drivers	P+D Dispatched per Driver
UAE B2C/Mixed	2018-04-09	8,565	211	40.6
KSA - Central Province	2018-04-09	6,130	217	28.2
KSA - Western Province	2018-04-09	3,326	169	19.7
UAE Special/Rak/Liv	2018-04-09	2,163	103	21.0
KSA - Eastern Province	2018-04-09	1,766	71	24.9
Bahrain	2018-04-09	575	18	31.9
Jordan	2018-04-09	267	16	16.7

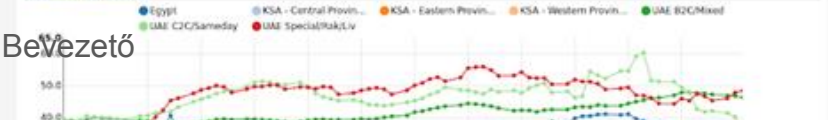
Topline Ops - P+D Delivered Per Drivers

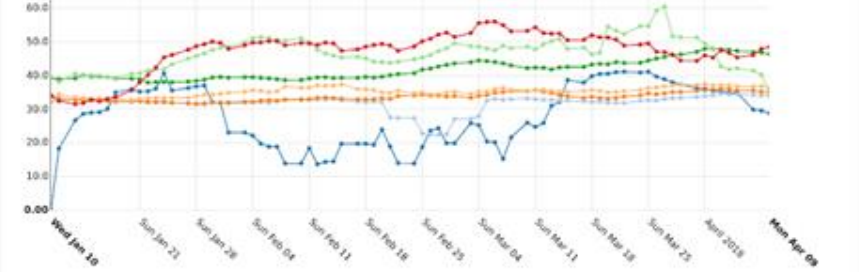


Topline Ops - Latest Delivered Per Driver

Region Fleet	_timestamp	P+D Deliveries	P+D Delivery %	P+D Delivery Drivers	P+D Deliveries per Driver
UAE B2C/Mixed	2018-04-09	6,942	81.1%	212	32.7
KSA - Central Province	2018-04-09	4,874	79.5%	216	22.6
KSA - Western Province	2018-04-09	2,696	81.1%	169	16.0
UAE Special/Rak/Liv	2018-04-09	2,075	95.9%	103	20.1
KSA - Eastern Province	2018-04-09	1,424	80.6%	71	20.1
Bahrain	2018-04-09	504	87.7%	18	28.0
Jordan	2018-04-09	212	79.4%	15	14.1

Topline Ops - FUD DPD

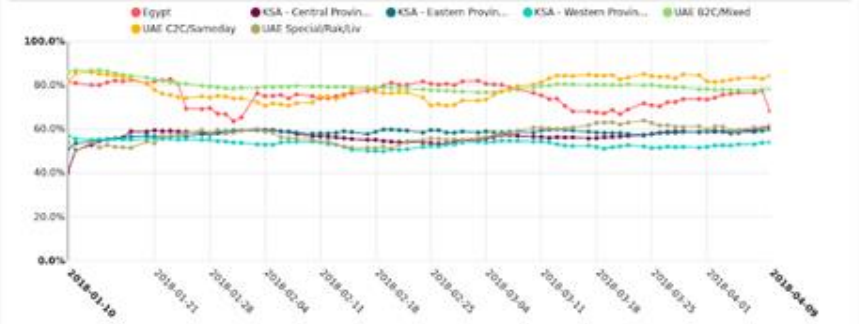




Topline Ops - Latest Fully Utilized Drivers

Region Fleet	_timestamp	Drivers	Fully Utilized Drivers	Fully Utilized Drivers %	FUD DPD
KSA - Central Province	2018-04-09	217	127	58.5%	37.0
UAE B2C/Mixed	2018-04-09	211	167	79.1%	45.9
KSA - Western Province	2018-04-09	169	24	14.2%	35.1
UAE Special/Rak/Liv	2018-04-09	103	23	22.3%	57.8
KSA - Eastern Province	2018-04-09	71	21	29.6%	33.1
UAE C2C/Sameday	2018-04-09	21	0	0.0%	0.00
Bahrain	2018-04-09	18	10	55.6%	44.3

Topline Ops - Deliveries on Time



Topline Ops - Latest Deliveries on Time

Region Fleet	_timestamp	Deliveries on Time %
UAE C2C/Sameday	2018-04-09	79.6%
UAE B2C/Mixed	2018-04-09	79.1%
Bahrain	2018-04-09	68.4%
Jordan	2018-04-09	66.3%
UAE Special/Rak/Liv	2018-04-09	63.8%
KSA - Central Province	2018-04-09	63.4%
KSA - Eastern Province	2018-04-09	62.1%

Számok, metrikák

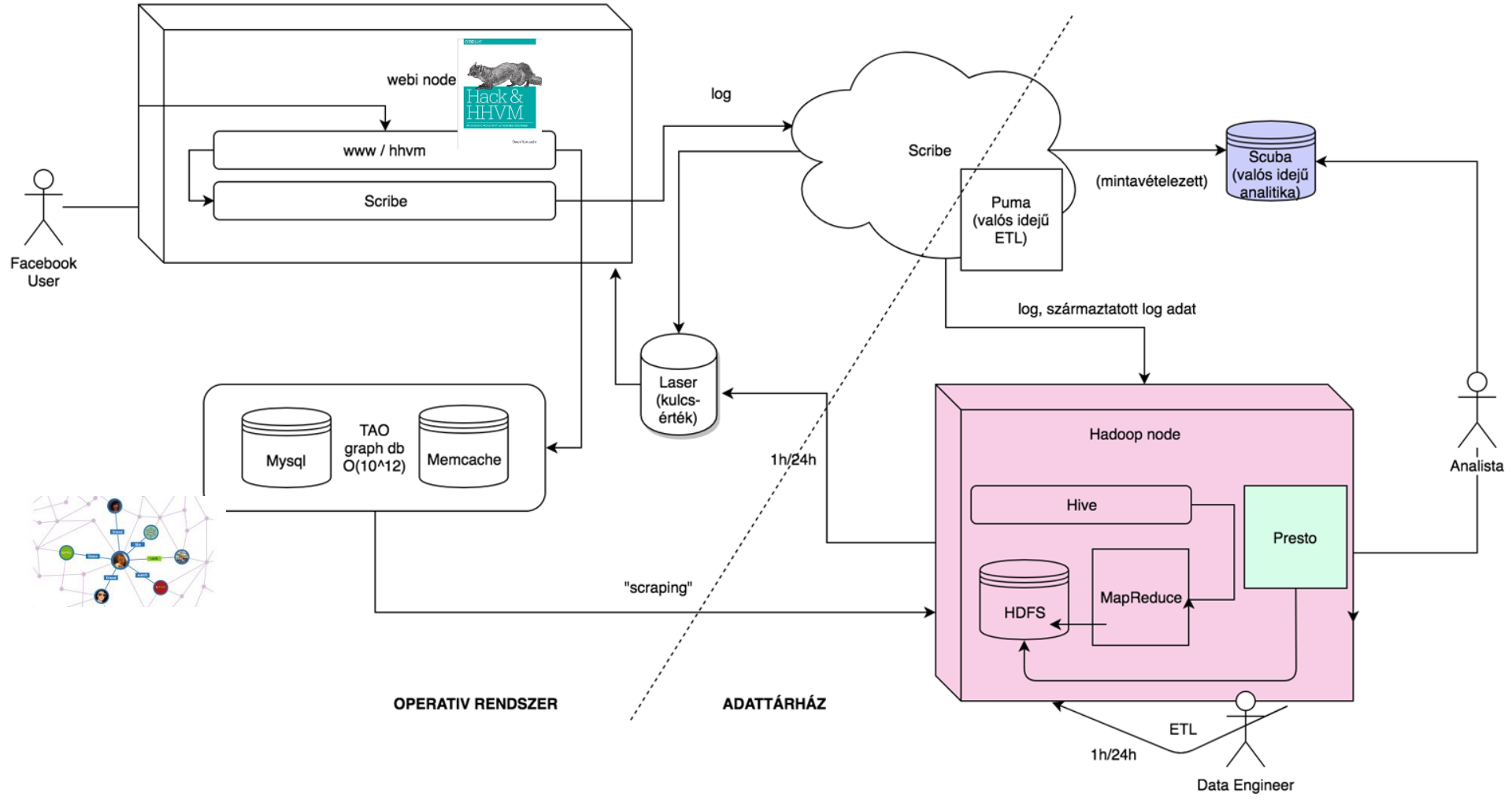
1. Daily Active User (DAU) - naponta aktív felhasználó
2. Monthly Active User (MAU) - utolsó 28 napban aktív felhasználó
3. Timespent / DAU - átlagosan hány percet tölt FB-on egy DAU

→ Mennyi adat generálódik naponta?

Legyen percenként 10KB → 1B x 60 x 10KB = 600TB naponta

De több termék van... FB, Messenger, Instagram, Whatsapp → több PB naponta
(kicsit segít: retention = letöröljük a régi, nem releváns DWH adatokat)

De: kommersz adatbázis kezelők nem boldogulnak ennyi adattal, vagy nem elég flexibilisek


webi node
Hack & HHVM
www / hhvm
Scribe
Facebook User
log
Scribe
Puma (valós idejű ETL)
(mintavételezett)
Scuba (valós idejű analitika)
log, származtatott log adat
Laser (kulcs-érték)
Mysql
TAO graph db O(10^12)
Memcache
1h/24h
Hadoop node
Hive
Presto
HDFS
MapReduce
Analista
"scraping"
OPERATIV RENDSZER
ADATTÁRHÁZ
ETL
1h/24h
Data Engineer

A megoldás (2008-ban) = Hadoop

Google-nél készültek hasonló rendszerek 2005 körül:

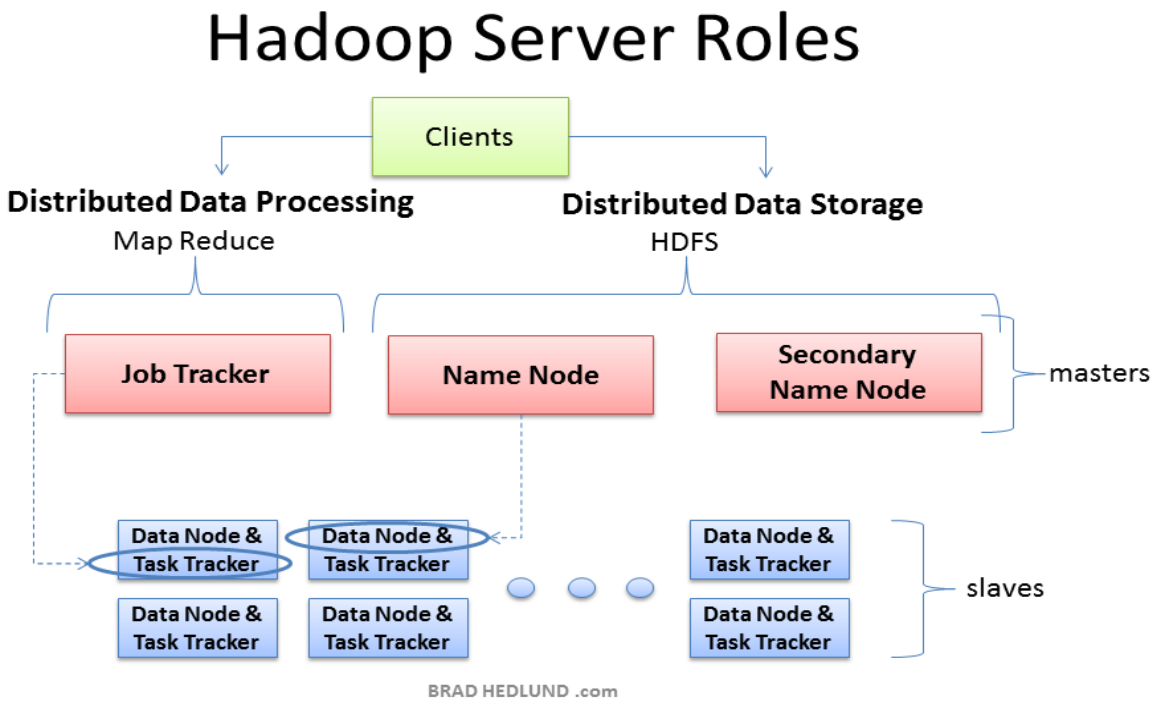
- Google Filesystem (GFS)
- Bigtable
- Chubby
- MapReduce

Ezekről cikkeket írtak a Google mérnökei, ez alapján a Yahoo! mérnökei elkészítettek egy hasonló architektúrájú **open-source** rendszert, ez a Hadoop (Java alapú).

Facebook-nál Hadoop 2008 óta van.

Hadoop klaszter

1. Storage: Hadoop Filesystem (HDFS)
   a. Name node
   b. Data node
2. Compute: MapReduce
   a. Job tracker
   b. Task tracker
3. Hive: SQL
   a. HCatalog
   b. HiveServer



HDFS tervezési elvek

1. Több ezer node, hardver failure mindig van.
2. Sok adat, PB-ok.
3. Write-once modell.
4. Nagy adatmennyiség miatt olcsóbb a számítást átmásolni mint az adatot.
5. Streaming jellegű adathozzáférés legyen gyors.

Namenode

A file rendszer metaadatot tárolja memóriában:

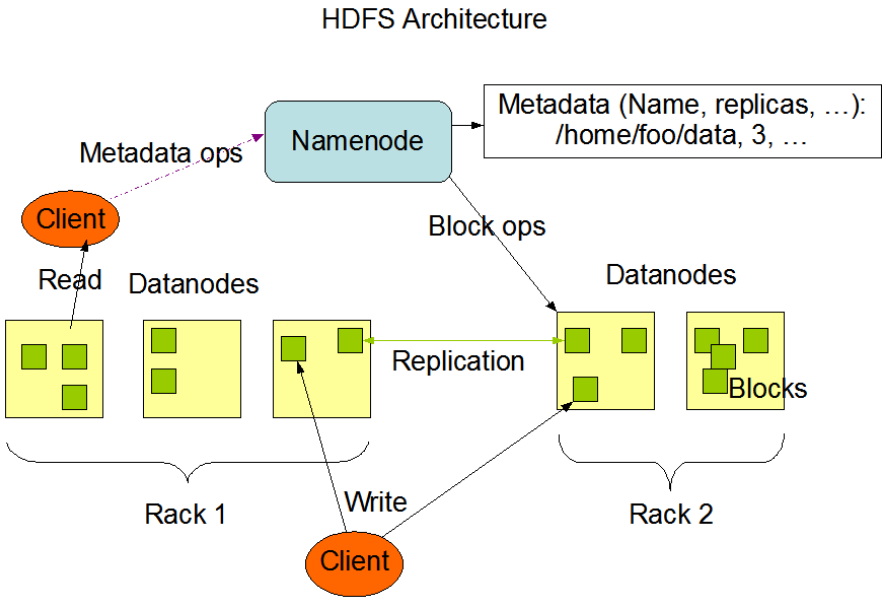
1. file-ok fába rendezése
2. egy file mely blokkokból áll (tipikusan egy blokk 128MB)
3. blokk → {Datanode} táblát
   R=3 (default) replikációs faktor miatt több Datanode
4. Datanode → {blokk} táblát

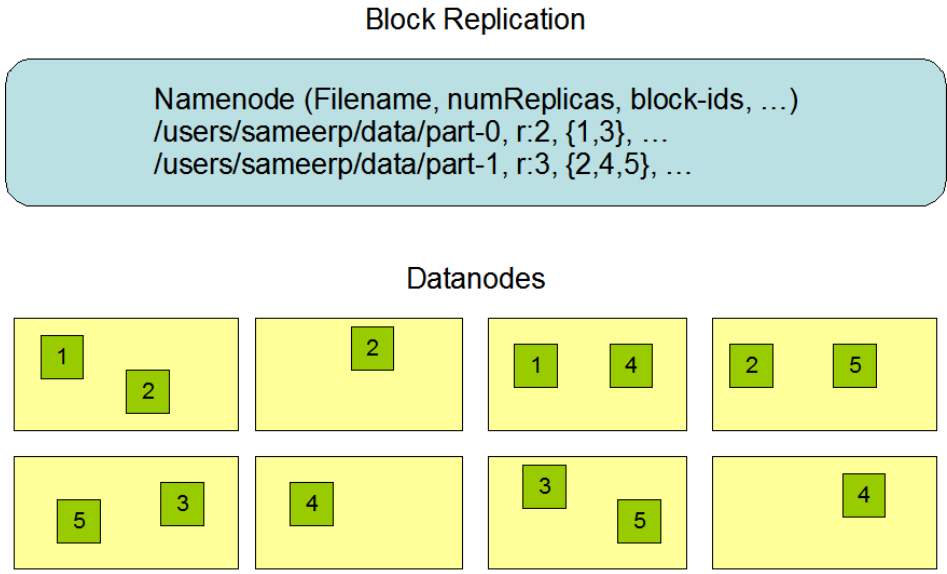
Hibatűrésre rendelkezésre áll egy **Secondary Namenode**, ami pár percenként másolja a Namenode-ot.

Datanode

1. 3 mp-enként heartbeat a Namenode-nak
2. A blokkjairól checksum-ot tart nyilván, ennek segítségével diszk korrupciót észleli, és jelenti a Namenode-nak
3. → ha szükség van rá, a Namenode újra replikál blokkokat amelyeknek az R' replikációs faktora R alá csökken.
4. Ennek hatására Datanode-ok egymás közt másolnak blokkokat.

HDFS


HDFS Architecture
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Client
Rack 2


Block Replication
Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …
Datanodes
1
2
2
1
4
2
5
5
3
4
3
5
4

Facebook HDFS

1. Több klaszter (silver, gold, stb.)
2. Egy klaszter = 5-10,000 node
3. Egy klaszteren belül több Hive namespace (mint séma az adatbázisoknál)
4. Termékenként (FB, Messenger, IG, WP) egy namespace.
5. Klaszterek között nehéz mozgatni adatot, de lehet.

“Friends viewing posts”

Nagy HDFS klaszter = nagyon sok adat, elosztva

Hogyan futtatunk rajta analitikai query-ket?

- Nem fér be memóriába
- Sokáig tart

```
SELECT
    v.post_id,
    COUNT(DISTINCT v.user_id) AS num_viewers
FROM
    views v
WHERE
    DATE(v.timestamp) = DATE('2018-04-01')
GROUP BY
    1
```

```
SELECT
    v.post_id,
    COUNT(DISTINCT v.user_id) AS num_viewers,
    COUNT(DISTINCT v2.user_id) AS num_viewers_with_viewer_friends
FROM
    views v
LEFT JOIN
    friends f
ON
    f.user_id1 = v.user_id
LEFT JOIN
    views v2
ON
    f.user_id2 = v2.user_id
WHERE
        (v.post_id = v2.post_id OR v2.post_id IS NULL)
    AND DATE(v.timestamp) = DATE('2018-04-01')
GROUP BY
    1
```

MapReduce segít

map(list, func) es **reduce(list, op, first)** a funkcionális programozásból

pl. `map([1, 2, 3], sin) → [sin(1), sin(2), sin(3)]`

pl. `reduce([1, 2, 3], operator+, 0) → 1+2+3 = 6`

MapReduce = elosztott programozási modell

- oldjuk meg a problémát egymás után következő map es reduce fázisokkal (map-reduce-reduce-map-map-reduce...)
- A fázisok közötti átmeneti eredményeket kiírja HDFS-re (→ nem kell sok memória), itt **shuffle** és **sort** lépések vannak

Példa

page_view

pageid	**userid**	time
1	**111**	9:08:01
2	**111**	9:08:13
1	**222**	9:08:14

X

user

userid	age	gender
111	25	female
222	32	male

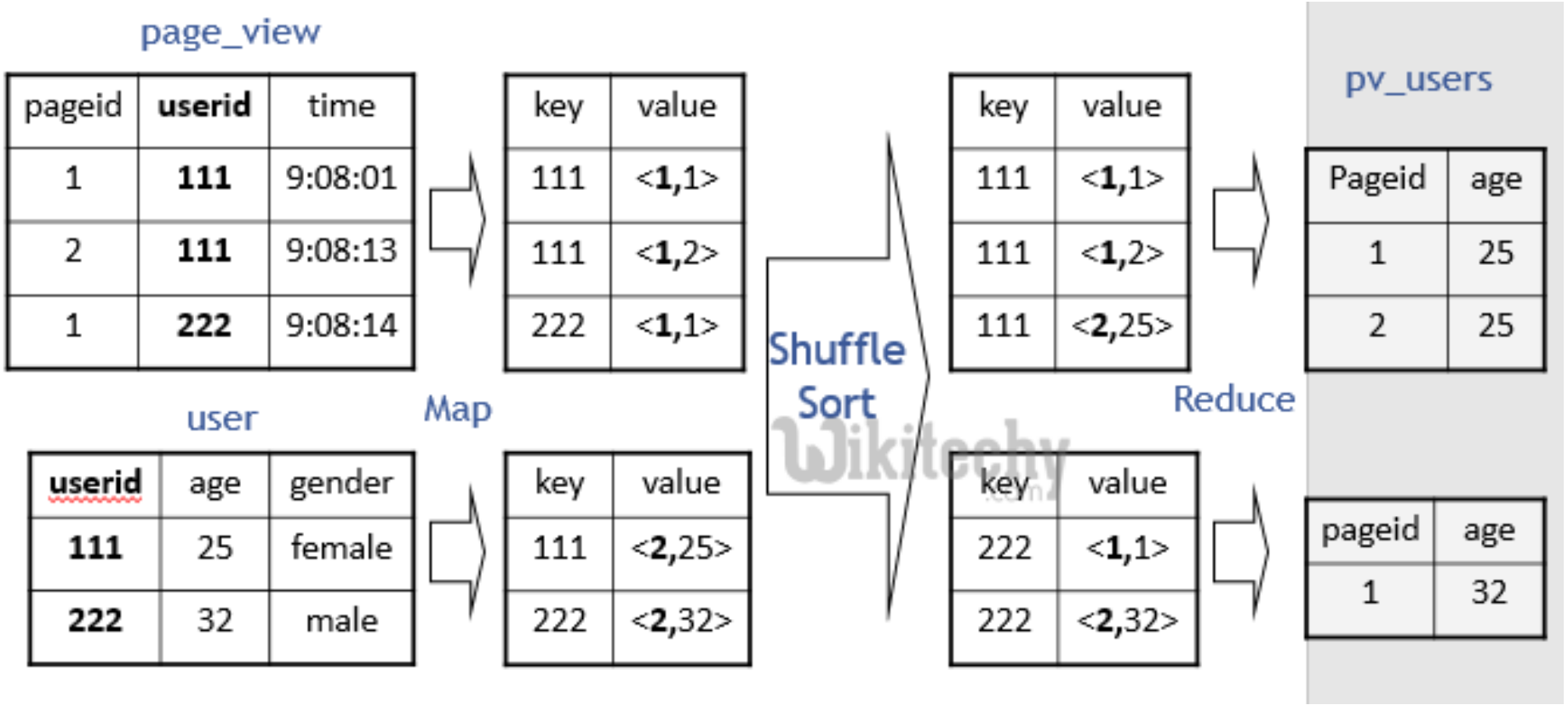
=

pv_users

pageid	age
1	25
2	25
1	32

- SQL:

```
INSERT INTO TABLE pv_users
SELECT pv.pageid, u.age
FROM page_view pv JOIN user u ON (pv.userid = u.userid);
```


page_view
pageid | userid | time
1 | 111 | 9:08:01
2 | 111 | 9:08:13
1 | 222 | 9:08:14
user
userid | age | gender
111 | 25 | female
222 | 32 | male
Map
key | value
111 | <1,1>
111 | <1,2>
222 | <1,1>
key | value
111 | <2,25>
222 | <2,32>
Shuffle
Sort
key | value
111 | <1,1>
111 | <1,2>
111 | <2,25>
key | value
222 | <1,1>
222 | <2,32>
Reduce
pv_users
Pageid | age
1 | 25
2 | 25
pageid | age
1 | 32

MapReduce 2 szerepből áll

1. **JobTracker**
   a. Átveszi a MR programot a klienstől
   b. Általában külön gépen fut, MR szempontból szűk keresztmetszet (single point of failure)
   c. NameNode-dal beszél, oda rakja az MR taskokat ahol az adat van (adatlokalitás)
   d. Figyeli az egyes TaskTrackerek-et, jelent a kliens-nek
   e. Ha lehal, MR nem megy, HDFS továbbra is megy
2. **TaskTracker**
   a. Map és Reduce végrehajtás
   b. DataNode-okon fut (adatlokalitás)
   c. JobTracker-nek állandóan jelent
   d. Ha lehal, nincs baj, a JobTracker újrafuttat egy másik TaskTracker-en

Hive

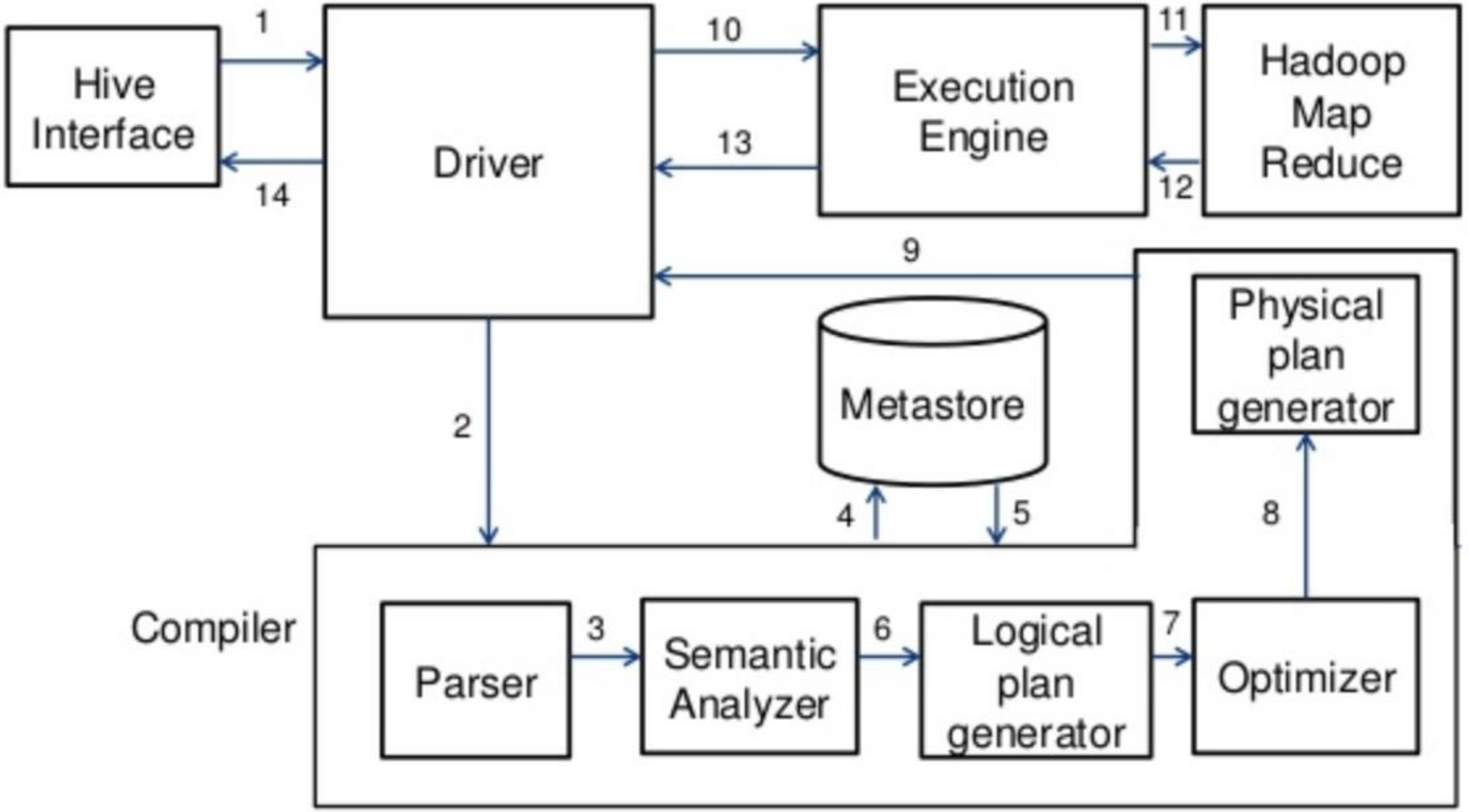
Lehetséges direktben MapReduce programot írni, tehat megadni a mapper (**sin**) és reducer (**operator+**) függvényeket egymás után, illetve a csomagoló (driver) programot. De ez még a FB-ban is ritka.

Helyette = Hive, azaz Hive QL (HQL)

SQL-szerű nyelv, amiből a Hive runtime legyártja a MapReduce programot

HQL query → HQL compiler → .jar MapReduce job

Két új szerep: **HCatalog** (séma, hol és hogyan), **HiveServer** (lekérdezések)

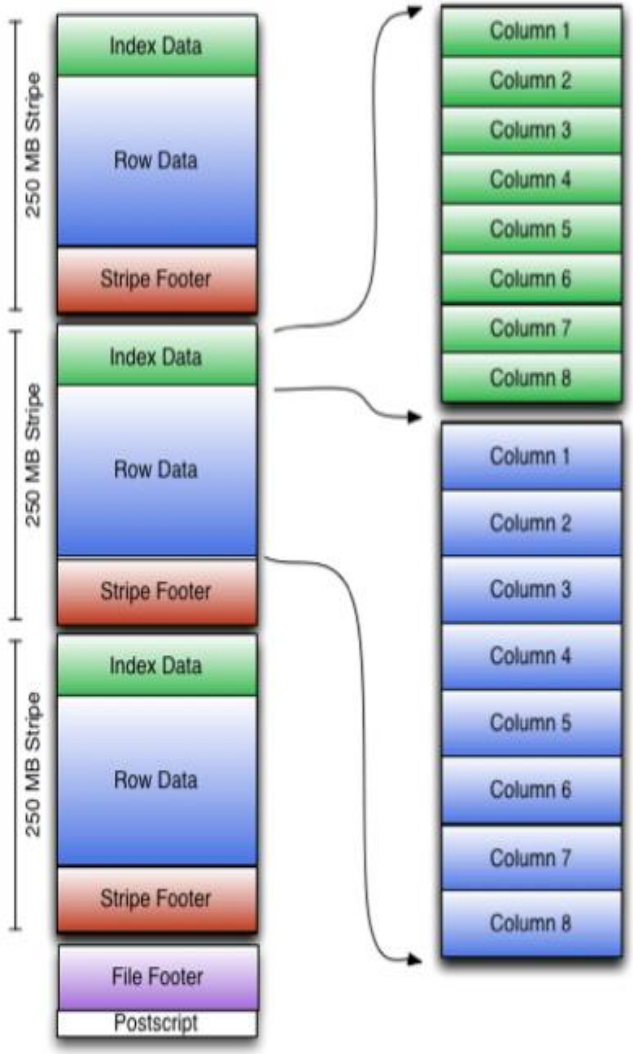

Hive Interface
Driver
Execution Engine
Hadoop Map Reduce
Metastore
Physical plan generator
Compiler
Parser
Semantic Analyzer
Logical plan generator
Optimizer
1
10
11
13
12
14
9
2
4
5
8
3
6
7

Hive táblák

File-ok a HDFS elosztott filerendszeren

hdfs://dwh/tablename/ds=2018-03-20/data.{csv,orc}

Lehet akár CSV file is,
vagy hatékonyabb formátumok, pl. FB-nál
Optimized Row Columnar (ORC) file.



Hive plusz és minuszok

Plusz:

- Sok adatra működik
- Hibatűrés, replikáció, nagy rendelkezésre állás (HDFS+MR)
- Viszonylag teljes SQL, UDF
- Moduláris, cserélhető komponensek (filerendszer, formátum, végrehajtás)
- Opensource

Nem:

- (Nem ANSI SQL)
- Nem operatív adatbázis
- és...

Probléma

Hive-nál mindenképpen legyárt egy MapReduce programot, mindent elosztottan, batch-ben csinál, mindig ki kell várni egy nagy/hosszú soros folyamatot, ahol sok diszk írás is történik, nincs okos caching beleépítve.

Pl.

```
SELECT 1
```

vagy

```
SELECT * FROM users LIMIT 10
```

is percekig tart :(

Nem lehetne “kisebb” query-kre egy gyorsabb SQL motort használni?

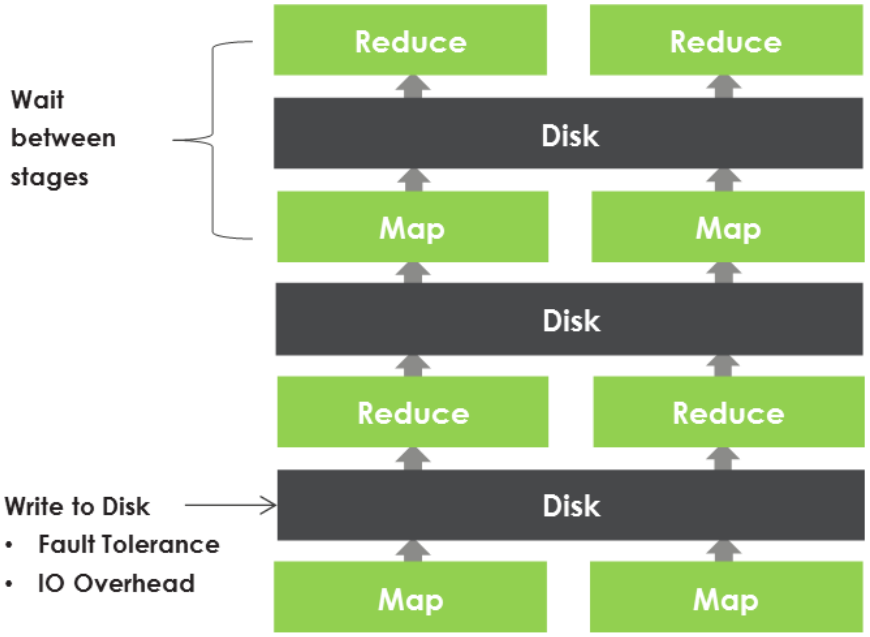
Presto

FB-ban írták, opensource, 2013-tól fut.

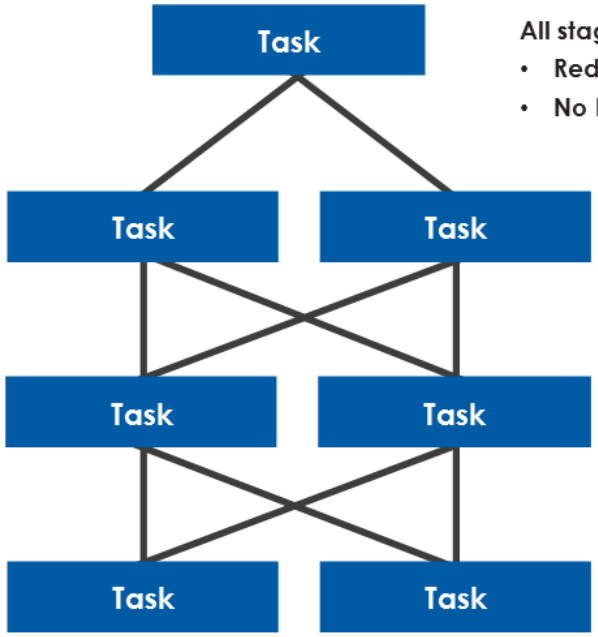
Ugyanazon HDFS elosztott klaszter felett ül, mint a Hive, de egy teljesen más, in-memory SQL execution engine (CSAK, DDL Hive-ban). Ha nem elég a memória, akkor ERROR, és át kell vinni a query-t Hive-ba.
Nagyon gyors:

- csak memóriában
- workerek egymásnak streamelnek
- csak oszlop alapú
- nagyon gyors bytecode-ot generál
- kis query-k nagyobb prioritást kapnak
- nincs fault-tolerance

Hive



Presto

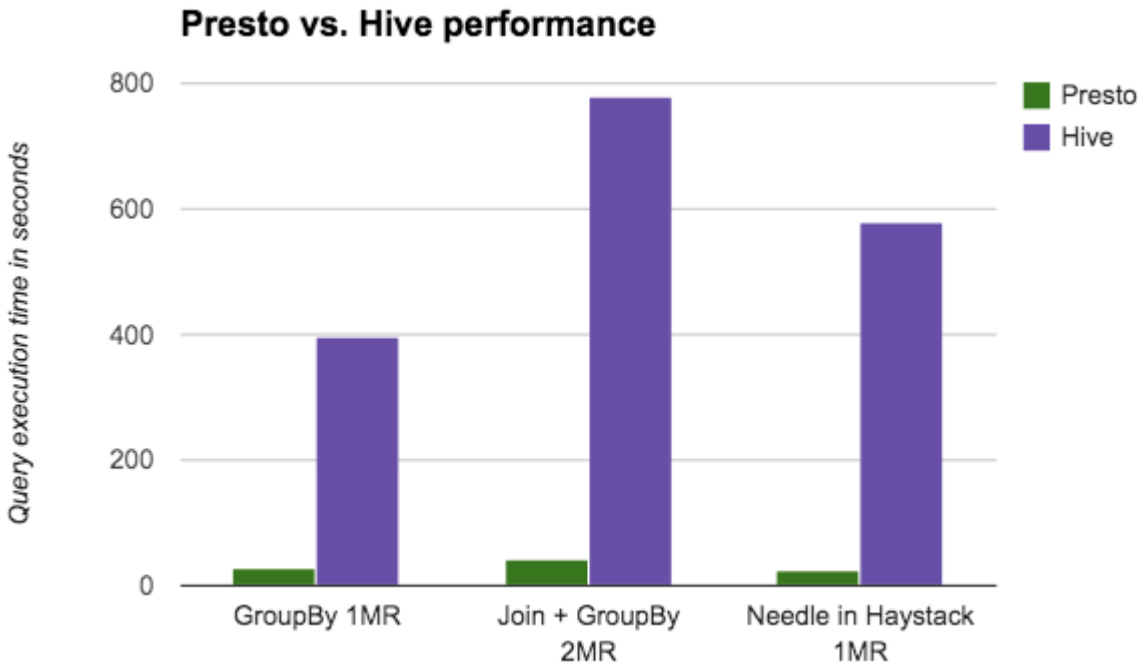


All stages are pipelined
- Reduced wait time
- No Fault Tolerance

Memory-to-memory Data transfer
- No disc IO
- Data chunk must fit in memory

TERADATA.

Tényleg :)


Presto vs. Hive performance
Query execution time in seconds
800
600
400
200
0
GroupBy 1MR
Join + GroupBy 2MR
Needle in Haystack 1MR
Presto
Hive

Presto vs Hive

Melyiket, mikor?

1. Facebook-ban eredetileg Hive volt, ezért a kb. 10,000+ ETL job nagy része **HQL** `INSERT INTO ... SELECT ... FROM` -ekből áll.
2. Új ETL jobokat már Presto-ban írnak, feltéve hogy lefut, egyébként visszaesnek Hive-ra.
3. A fontosabb jobokat migrálják Presto-ra (és Spark-ra).
4. Az analisták (“Data Scientist”) Presto lekérdezéseket írnak egész nap.
    a. Ha túl nagy → Hive :(

Event monitoring

"Former Facebook employee and current Googler. I really really miss Scuba."

1. Nagy mennyiségű log adat, percenként milliárdos nagyságrendű eventszám.
   `{ts: 1523029136, post_id: 1360582939, user_id:8502858292, length_msec: 600}`
2. Monitoring, mind infra-ra, mind a termékre.
   Pl.
   “kiraktunk egy új signup page-t, be tudnak jelentkezni az emberek? Magyarországon is, vagy elfelejtettük megcsinálni a lokalizációt?”
3. → 1 percen belül kell a válasz, különben több százmillió ember nem tud belépni.
4. Viszont, nem kellenek 100% pontosságú válaszok, elég ha látjuk h “igen, be tudnak jelentkezni”.

Scuba, 2011

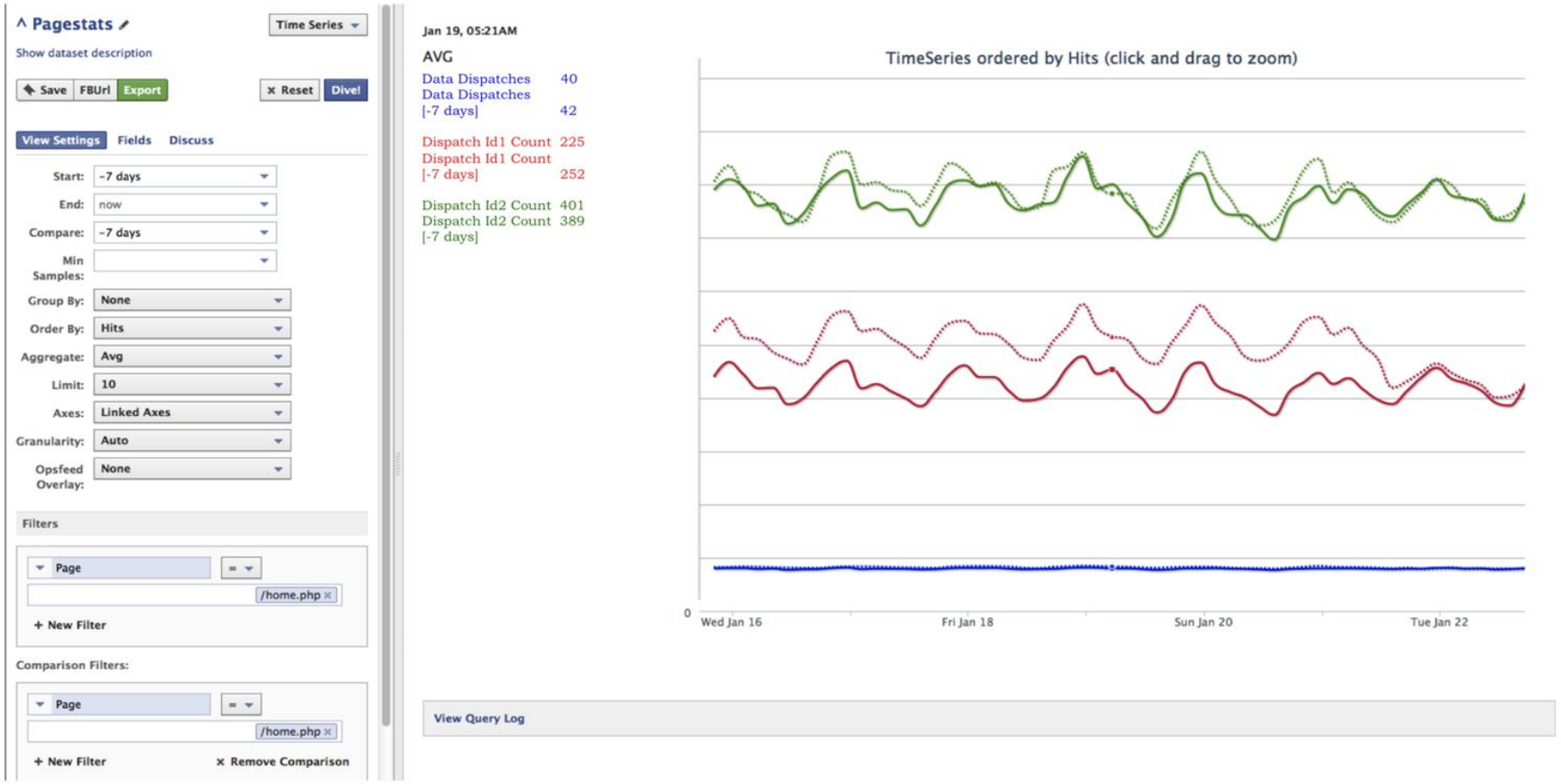


Figure 1: Scuba's web user interface. The query shown on the left side generates a time series graph with a week over week comparison of three columns related to Facebook page dispatches. The dotted lines represent the same days one week earlier. It is very easy to see daily and weekly cyclical behavior with these graphs.

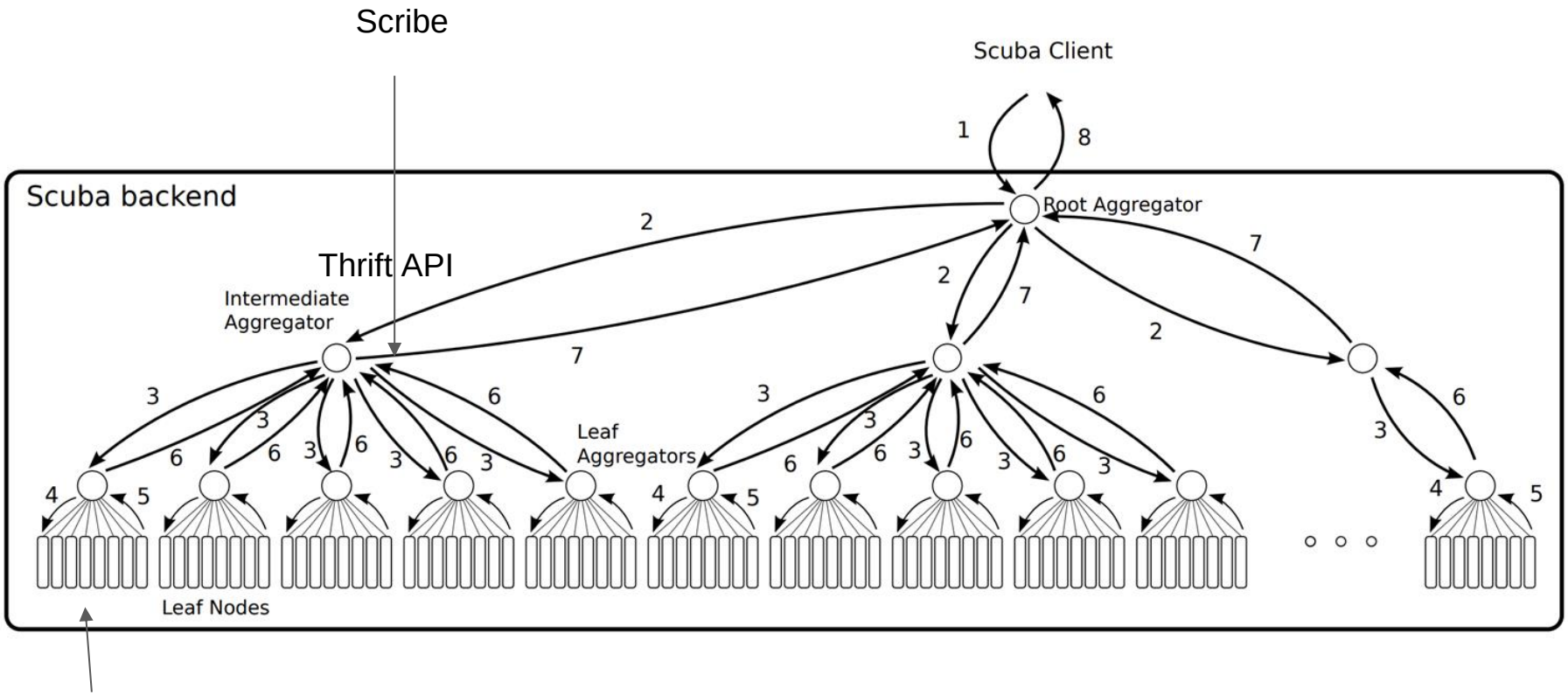
Adat betöltés

1. Nincsen definiált séma, a leaf node-ok direktben kapják az adatot a Scribe stream-ekből, egy Thrift API-n keresztül.
2. Minden blokk beérkező log sor esetén kiválaszt két véletlen Leaf Node-t, és az egyikre, amelyiken több szabad memória van, rakja az adatot.
3. Tehát az adatok, log kategória / táblától függetlenül minden Leaf Node-on szét vannak szórva. **Replikáció nincs.**
4. Minden adat log adat es az első mező a timestamp (index).
5. Nincs INSERT/UPDATE/DELETE, csak a fenti módon kerül be adat Scuba-ba.
6. Törlés: kor és tábla méret alapján (tipikus, 30 nap és 100GB ablak), ezen túl a legrégibb adatot ejti. Közben gzipelve diszken tárolja, de alapvetően memória alapú adatbázis (1 node 144GB), pár 1000 node.

Lekérdezés

1. Minden lekérdezés timestamp ablakra fut, nincs JOIN, nincs SUB-SELECT.
2. SELECT ... WHERE ... GROUP BY ... ORDER BY ... LIMIT.
3. Root Aggregator Node átveszi a query-t, leellenőrzi. AVG → SUM+COUNT
4. Kiválaszt +4 Intermediate Aggregator Node-ot, és átadja nekik a lekérdezést.
5. Kiválaszt +4 Intermediate Aggregator Node-ot, és átadja nekik a lekérdezést.
6. ...
7. Végül leérünk a Leaf Node-okhoz, azok kiszámolják a lokális adatokon.
8. Felfele progagál az eredmény, minden Aggregator kiszámolja az aggregátumokat.
9. Ha egy Lead Node nem válaszol 10 ms-on belül, kimarad.
10. Index csak timestamp-re van.
11. Minden szerveren futnak Leaf Node-ok és egy Aggregator Node.

Architektúra



1 Leaf Node / CPU core Minden Lead Node-on van minden táblának adata!

Trükkök

1. Idő granularitás miatt (pl. óránként mutasd a görbét), majdnem mindig van GROUP BY.
2. Hogy lesz így AVG()? SUM() + COUNT()
3. Hogy lesz LIMIT? *Best-effort:* MAX(5 ∗ limit, 100)
4. Ha egy oszlop nincs ott mert megváltozott a loggolás → NULL.
5. Legnagyobb trükk: a bejövő események mintavételezve vannak (még a Scuba előtt), minden sorban van egy speciális **sample_rate**=1...1,000,000 mező, ezt figyelembe veszi a SCUBA pl. COUNT() vagy SUM() számításnál!
6. Csak arra lehet szűrni, ami benne van a log sorban, pl. “country”, hiszen nincsen JOIN. **Tehát “kövér” logsorokat kell beküldeni Scuba-ba!**

Használat

- SLA UI:
  - 1 másodpercen belül visszajön az eredmény
  - az adatok >99% bele van számolva
- Monitoring
  - Ha egy görbe beesik ⟶ alert a megfelelő csapatnak

Tervezési elvek

- Iteratív, lehetőleg apró változások, kerüljük el az “újraírást”
- Építsünk a létező infrastruktúrára
- Jó mérnöki kompromisszumok (pl. csak in-memory)
- Olcsóbb sok memóriát venni mint megoldani hogy kevesebb memóriával, diszken is működjön (ma egy 12 core + 192GB RAM Dell szerver ~$1000, ez kb. 1-2 napi bére egy FB fejlesztőnek)
- Felhasználó kihasználása (Hive, Presto)
- Mi az ami elég? (Scuba)
- Bontsuk szét a problémát (Puma, Scuba)
- “Csalás”: ott dolgozik aki írta az adatbázis szoftvert...

Záró gondolatok

Hol érhető tetten a Facebook (user-adoption és üzleti) sikere a DWH architektúrában, annak környékén?

- Hive túl lassú, de ne migráljunk → írjunk egy execution engine-t ami ugyanaz az adat (HDFS) felett fut (zseniális, hasonló történt PHP/Hack téren is)
- Scuba → nem kell precizitás, majdnem mindig az adat 99% alapján is meg tudjuk hozni a döntéseinket
- Legnagyobb érték a termékfejlesztési sebesség (velocity) → minél jobban változik a termék/csapatok, annál jobb → nincs idő starschema-ra, minden tábla “lapos”, inkább megoldjuk sok hardverrel
- mi történt 30-90-180 napnál régebben ritkán számít → retention

Kérdések (és válaszok)