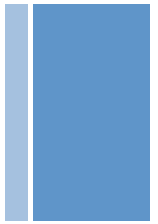
Objetivo del curso



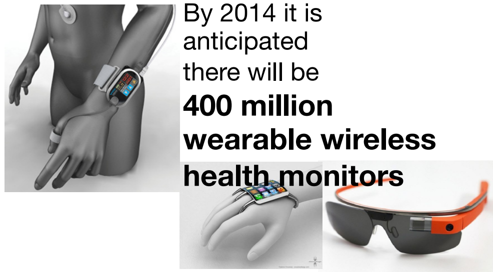
- Presentar los retos asociados al proceso de Big data y comprender y aplicar técnicas de procesamiento y análisis à colecciones de datos
- **Clases** (7): 15 horas
  - 19-21, 24 – 27 noviembre 2014
- **Proyecto de curso**: front end tipo open data market de colección de datos que demuestre 3 fases del ciclo de vida de big data

Data sharding and replication

Genoveva Vargas Solar
French Council of scientific research, LIG-LAFMIA, France
Genoveva.Vargas@imag.fr
http://www.vargas-solar.com

Data all around

As of 2011 the global size
Data in healthcare was estimated to be
150 Exabytes
(161 billion of Gigabytes)



By 2014 it is anticipated there will be
400 million wearable wireless health monitors

Web 2.0 sites where millions of users may both read and write data, scalability for simple database operations has become important

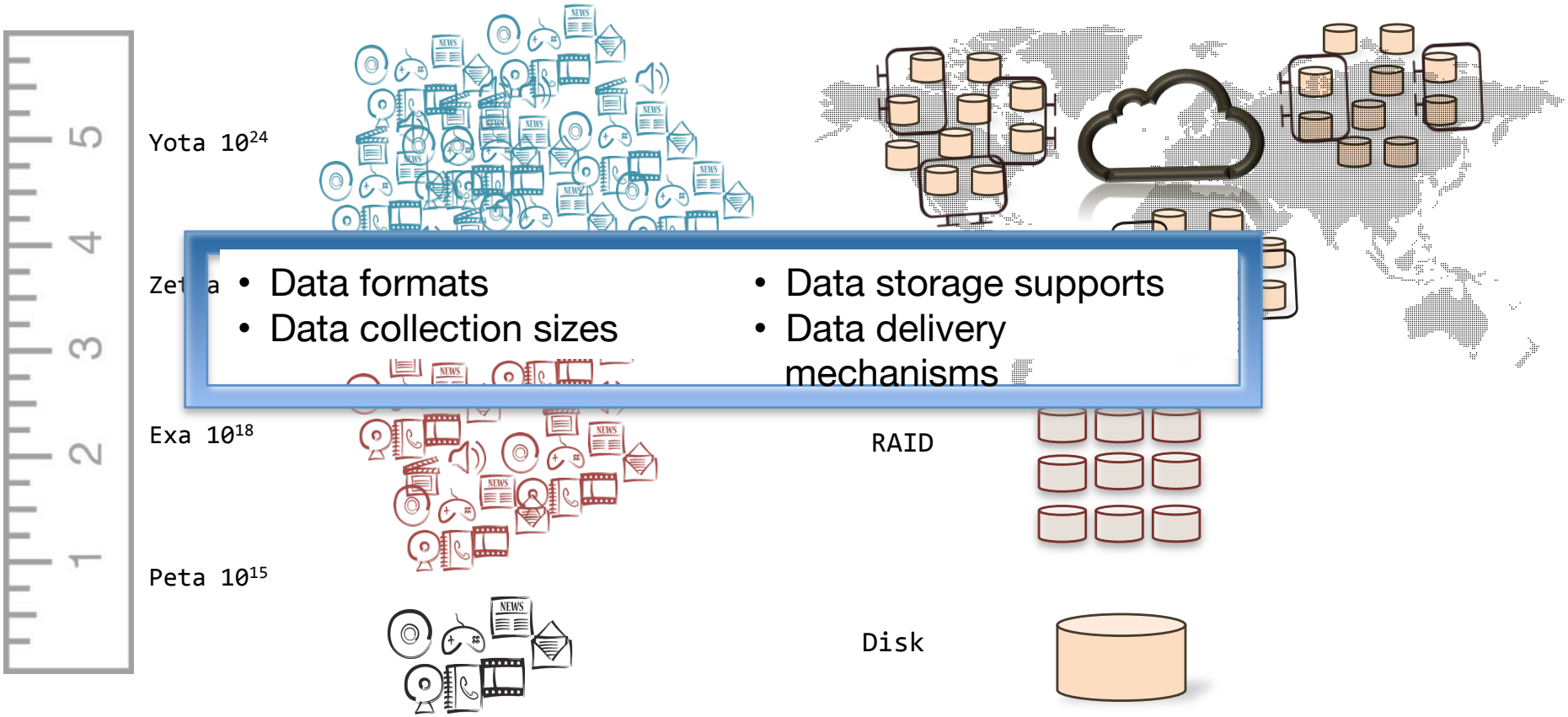
Data collections available through front ends managed through public/private organization available through the Web, e.g., Sloan Sky Server

40 billion+ hours video
are watched on YouTube each month

30 billion pieces of content
are shared on Facebook every month

40 million Tweets
are sent per day about 200
monthly active users

Storing and accessing huge amounts of data



This part of the course is about

NoSQL

Your Ultimate Guide to the Non-Relational Universe!

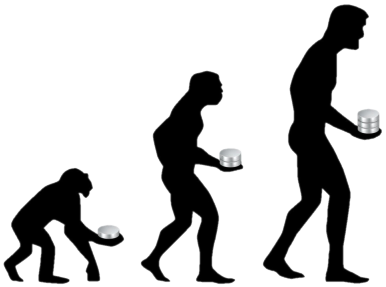
[including a **historic** Archive 2009-2011]
News Feed covering some changes here !

`http://nosql-database.org`

Debate on whether NoSQL stores and relational systems are better or worse …
that is not the point

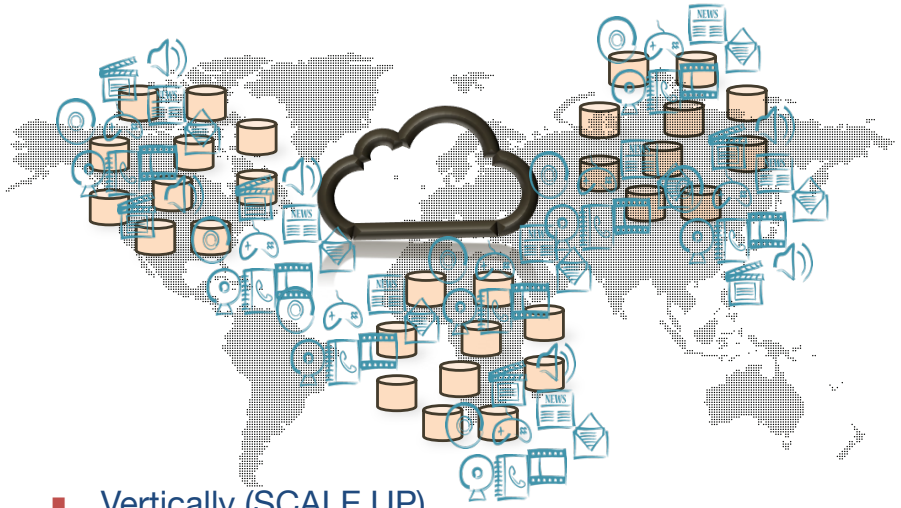
This session is absolutely about

Alternative for managing multiform and multimedia data collections *according to different properties and requirements*



Scaling database systems

- A system is scalable if increasing its resources (CPU, memory, disk) results in **increased performance** proportionally to the added resources
- Improving performance means serving more units of work for handling larger units of work like when data sets grow
- Database systems have been scaled by buying bigger faster and more expensive machines



- Vertically (SCALE UP)
  - Add resources (CPU, memory) to a single node in a system
- Horizontally (SCALE OUT)
  - Add more nodes to a system

NoSQL stores characteristics

There is no standard definition of what NoSQL means. The term began with a workshop organized in 2009, but there is much argument about what databases can truly be called NoSQL.

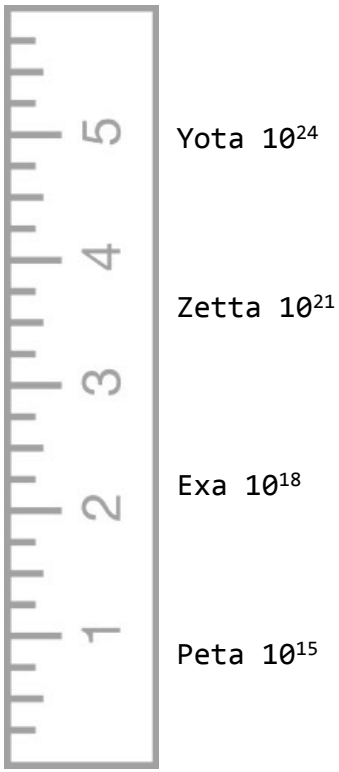
But while there is no formal definition, there are some common characteristics of NoSQL databases

- they don't use the relational data model, and thus don't use the SQL language
- they tend to be designed to run on a cluster
- they tend to be Open Source
- they don't have a fixed schema, allowing you to store any data in any record

- Simple operations
  - Key lookups reads and writes of one record or a small number of records
  - No complex queries or joins
  - Ability to dynamically add new attributes to data records
- Horizontal scalability
  - Distribute data and operations over many servers
  - Replicate and distribute data over many servers
  - No shared memory or disk
- High performance
  - Efficient use of distributed indexes and RAM for data storage
  - Weak consistency model
  - Limited transactions

Next generation databases mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontally scalable** `[http://nosql-database.org]`

Dealing with huge amounts of data

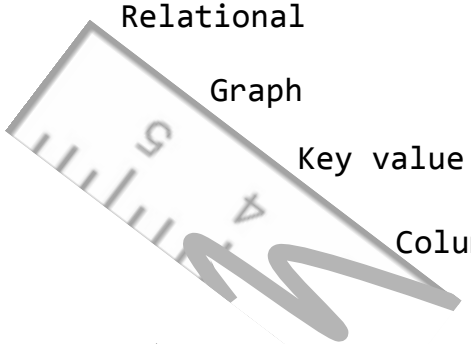


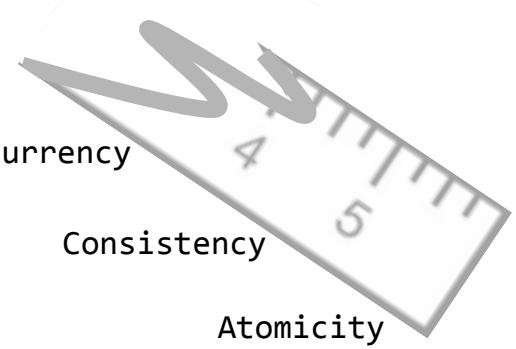
Yota 10^{24}

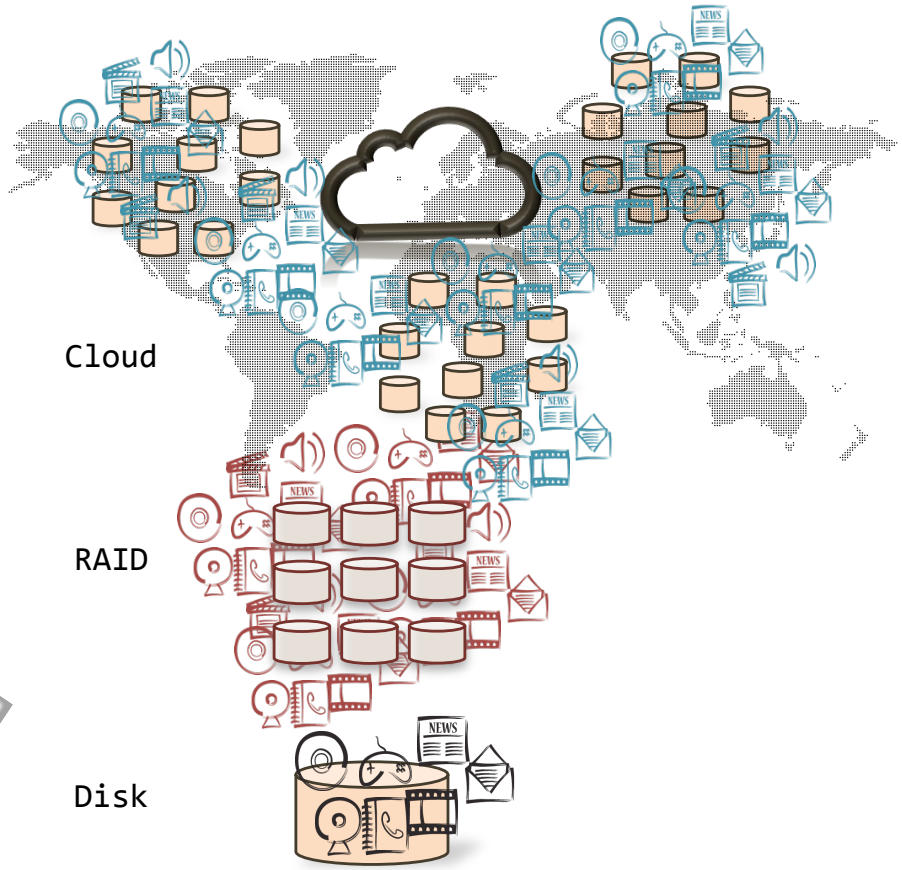
Zetta 10^{21}

Exa 10^{18}

Peta 10^{15}







so now we have NoSQL databases

https://www.youtube.com/watch?v=jyx8iP5tfCI

- Data model
- Consistency
- Storage
- Durability
- Availability
- Query support

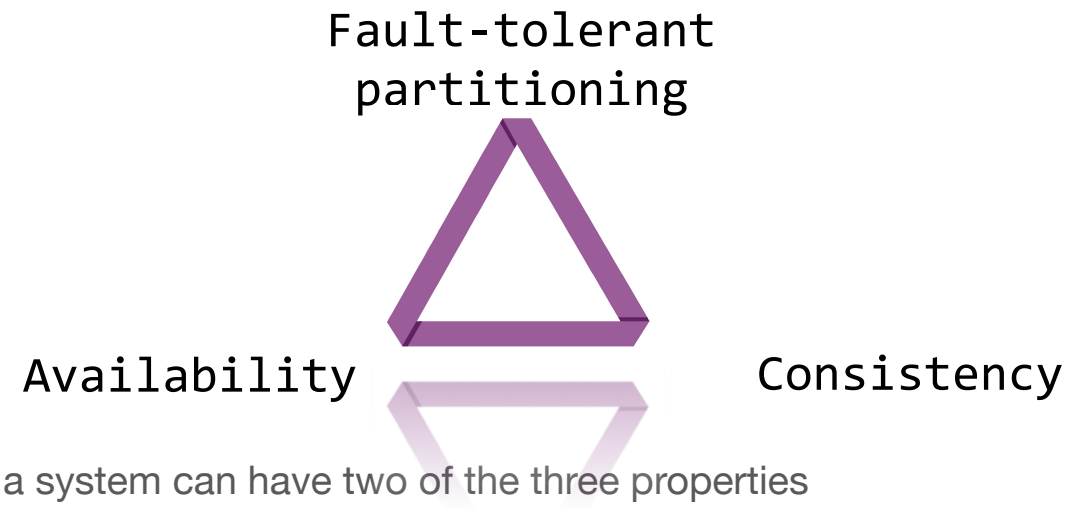
Data stores designed to scale simple OLTP-style application loads

Read/Write operations by **thousands/millions** of users

examples include

We should also remember Google's Bigtable and Amazon's SimpleDB. While these are tied to their host's cloud service, they certainly fit the general operating characteristics

Problem statement: How much to give up?

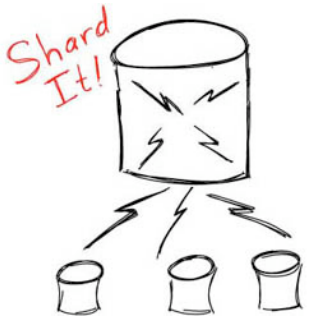


- CAP theorem[1]: a system can have two of the three properties
- NoSQL systems sacrifice **consistency**

1 Eric Brewer, "Towards robust distributed systems." PODC. 2000 http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

NoSql Stores: availability and performance

- Replication
  - Copy data across multiple servers (each bit of data can be found in multiple servers)
  - Increase data availability
  - Faster query evaluation
- Sharding
  - Distribute different data across multiple servers
  - Each server acts as the single source of a data subset
- Orthogonal techniques

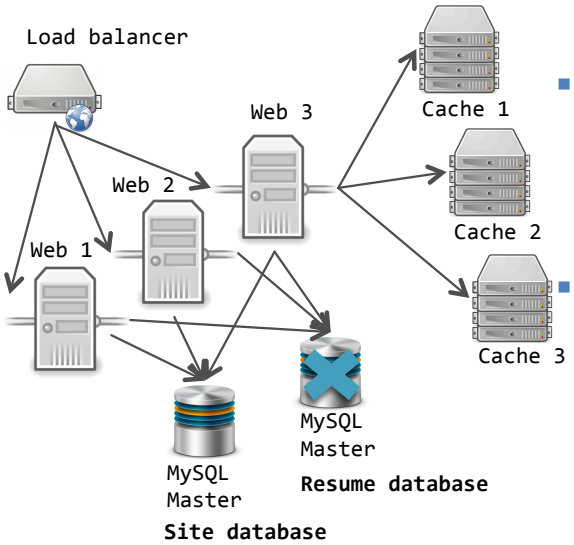


Replication: pros & cons

- Data is more available
  - Failure of a site containing E does not result in unavailability of E if replicas exist
- Performance
  - Parallelism: queries processed in parallel on several nodes
  - Reduce data transfer for local data
- Increased updates cost
  - Synchronisation: each replica must be updated
- Increased complexity of concurrency control
  - Concurrent updates to distinct replicas may lead to inconsistent data unless special concurrency control mechanisms are implemented

Sharding: why is it useful?

- Scaling applications by reducing data sets in any single databases
- Segregating data
- Sharing application data
- Securing sensitive data by isolating it



- Improve read and write performance
  - Smaller amount of data in each user group implies faster querying
  - Isolating data into smaller shards accessed data is more likely to stay on cache
  - More write bandwidth: writing can be done in parallel
  - Smaller data sets are easier to backup, restore and manage
- Massively work done
  - Parallel work: scale out across more nodes
  - Parallel backend: handling higher user loads
  - Share nothing: very few bottlenecks
- Decrease resilience improve availability
  - If a box goes down others still operate
  - But: Part of the data missing

Sharding and replication

- Sharding with no replication: unique copy, distributed data sets
  - (+) Better concurrency levels (shards are accessed independently)
  - (-) Cost of checking constraints, rebuilding aggregates
  - Ensure that queries and updates are distributed across shards
- Replication of shards
  - (+) Query performance (availability)
  - (-) Cost of updating, of checking constraints, complexity of concurrency control
- Partial replication (most of the times)
  - Only some shards are duplicated

NoSQL STORES: Data management properties

- Indexing
  - Distributed hashing like *Memcached* open source cache
    - In-memory indexes are scalable when distributing and replicating objects over multiple nodes
  - Partitioned tables
- High availability and scalability: eventual consistency
  - Data fetched are not guaranteed to be up-to-date
  - Updates are guaranteed to be propagated to all nodes eventually
- Shared nothing horizontal scaling
  - Replicating and partitioning data over many servers
  - Support large number of simple read/write operations per second (OLTP)
- No ACID guarantees
  - Updates eventually propagated but limited guarantees on reads consistency
  - BASE: basically available; soft state, eventually consistent
  - Multi-version concurrency control

Key-Value

Document

System	Concurrency control	Data storage	Replication	Transaction
Redis	Locks	RAM	Asynchronous	No
Scalaris	Locks	RAM	Synchronous	Local
Tokyo	Locks	RAM/Disk	Asynchronous	Local
Voldemort	MVCC	RAM/BDB	Asynchronous	No
Riak	MVCC	Plug in	Asynchronous	No
Membrain	Locks	Flash +Disk	Synchronous	Local
Membase	Locks	Disk	Synchronous	Local
Dynamo	MVCC	Plug in	Asynchronous	No
SimpleDB	Non	S3	Asynchronous	No
MongoDB	Locks	Disk	Asynchronous	No
CouchDB	MVCC	Disk	Asynchronous	No

Extended records

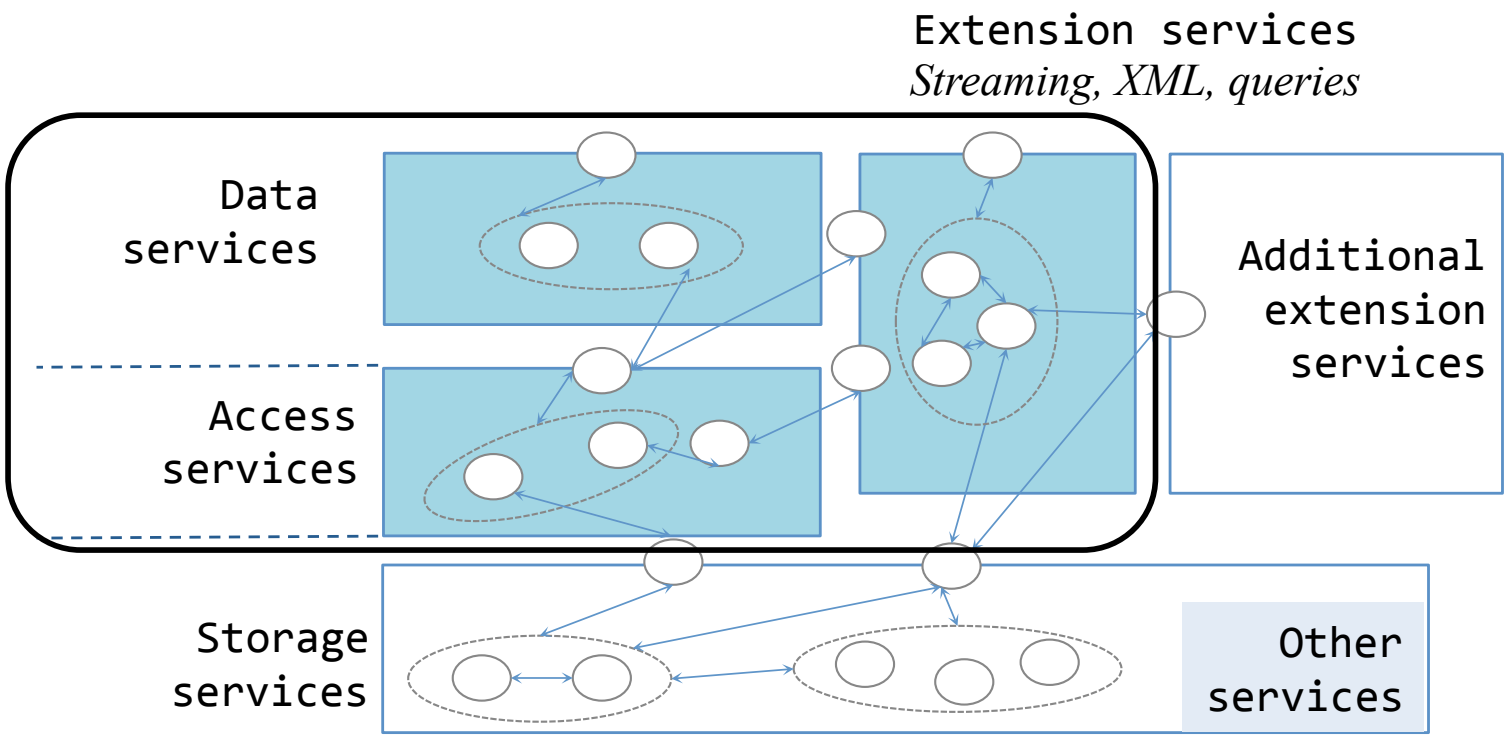
Relational

System	Concurrency control	Data storage	Replication	Transaction
Terrastore	Locks	RAM+	Synchronous	L
Hbase	Locks	HADOOP	Asynchronous	L
HyperTable	Locks	Files	Synchronous	L
Cassandra	MVCC	Disk	Asynchronous	L
BigTable	Locs +stamps	GFS	Both	L
PNuts	MVCC	Disk	Asynchronous	L
MySQL-C	ACID	Disk	Synchronous	Y
VoltDB	ACID/no Lock	RAM	Synchronous	Y
Clustrix	ACID/no Lock	Disk	Synchronous	Y
ScaleDB	ACID	Disk	Synchronous	Y
ScaleBase	ACID	Disk	Asynchronous	Y
NimbusDB	ACID/no Lock	Disk	Synchronous	Y

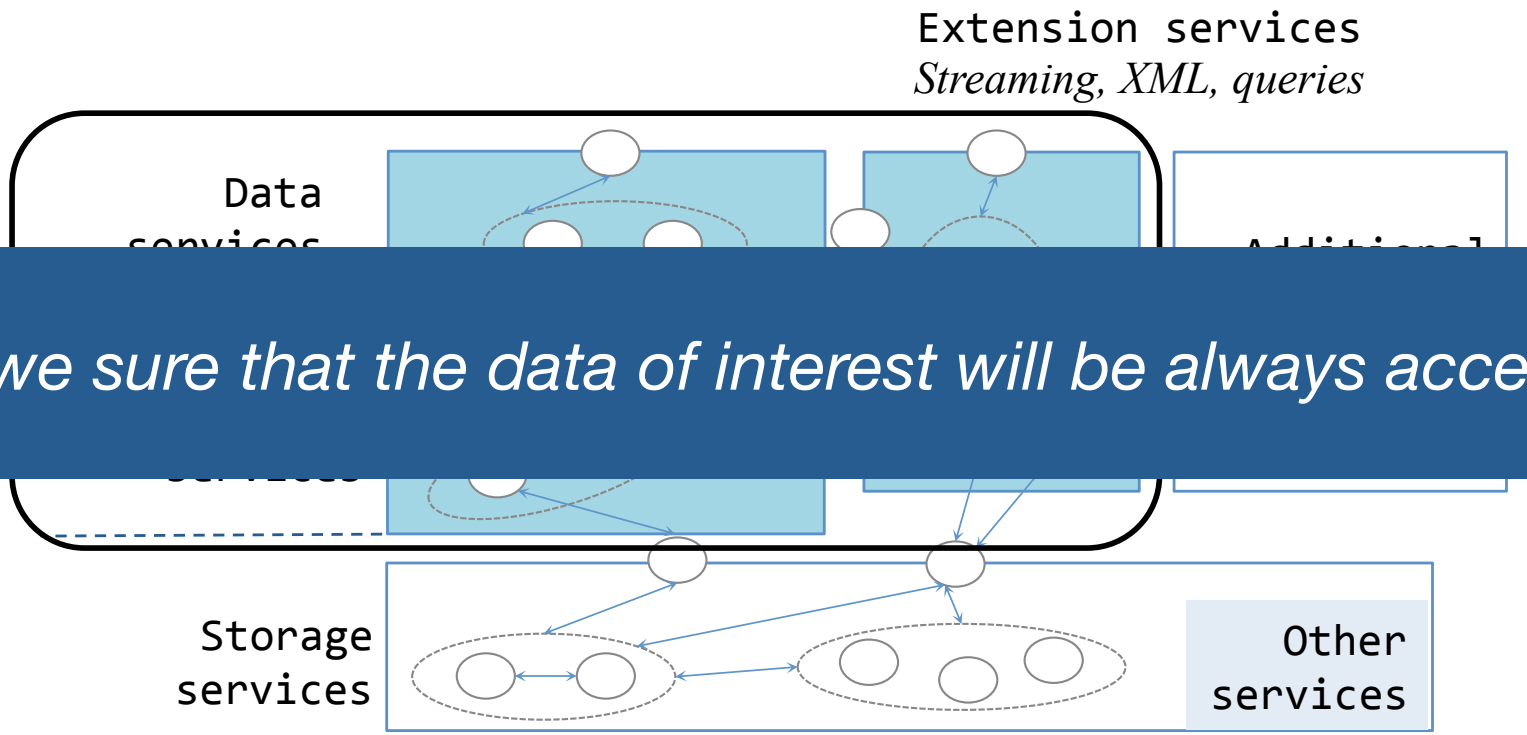
Cattell, Rick. "Scalable SQL and NoSQL data stores." ACM SIGMOD Record 39.4 (2011): 12-27

Tailoring data storage services

Service based HQ evaluation

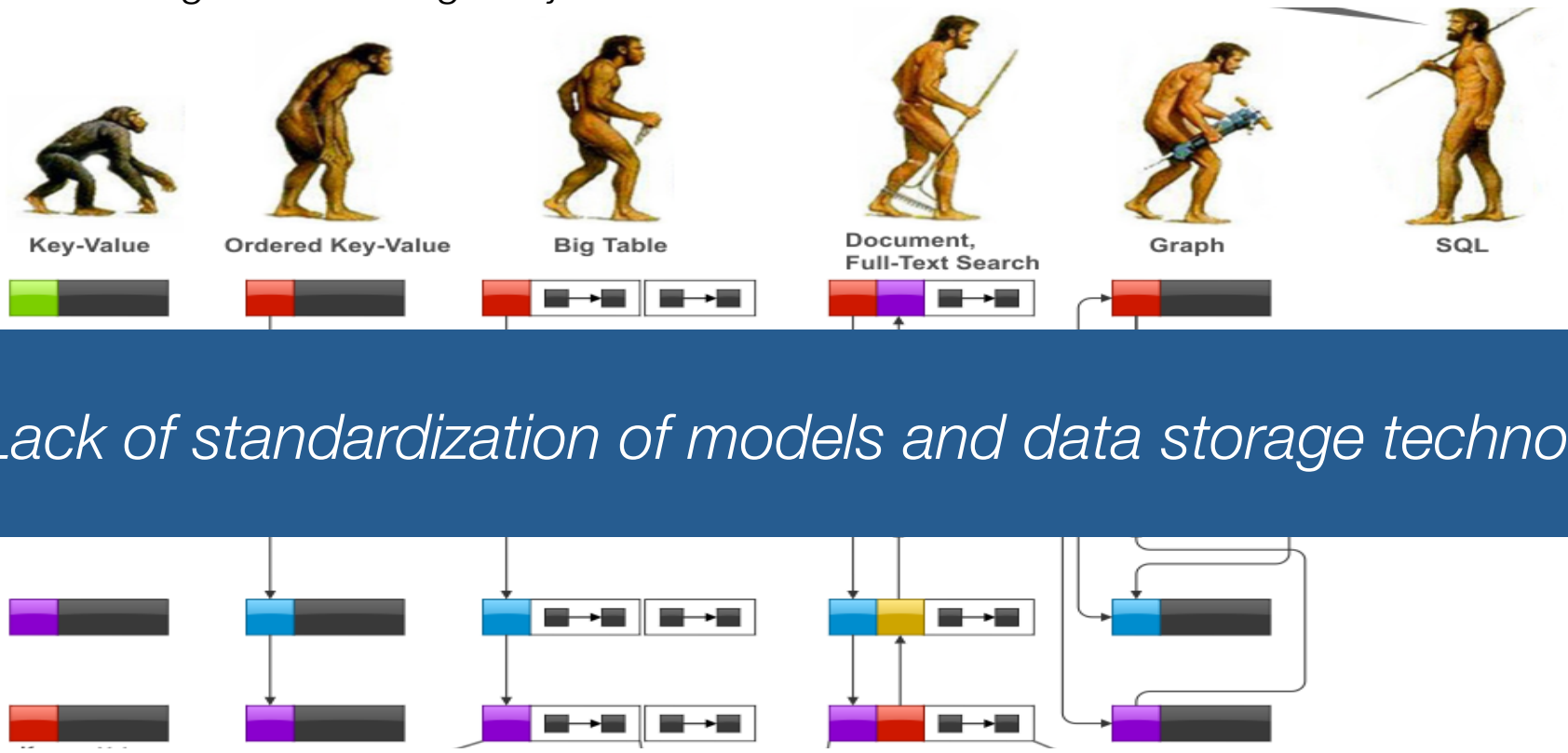


Service based HQ evaluation



Are we sure that the data of interest will be always accessible?

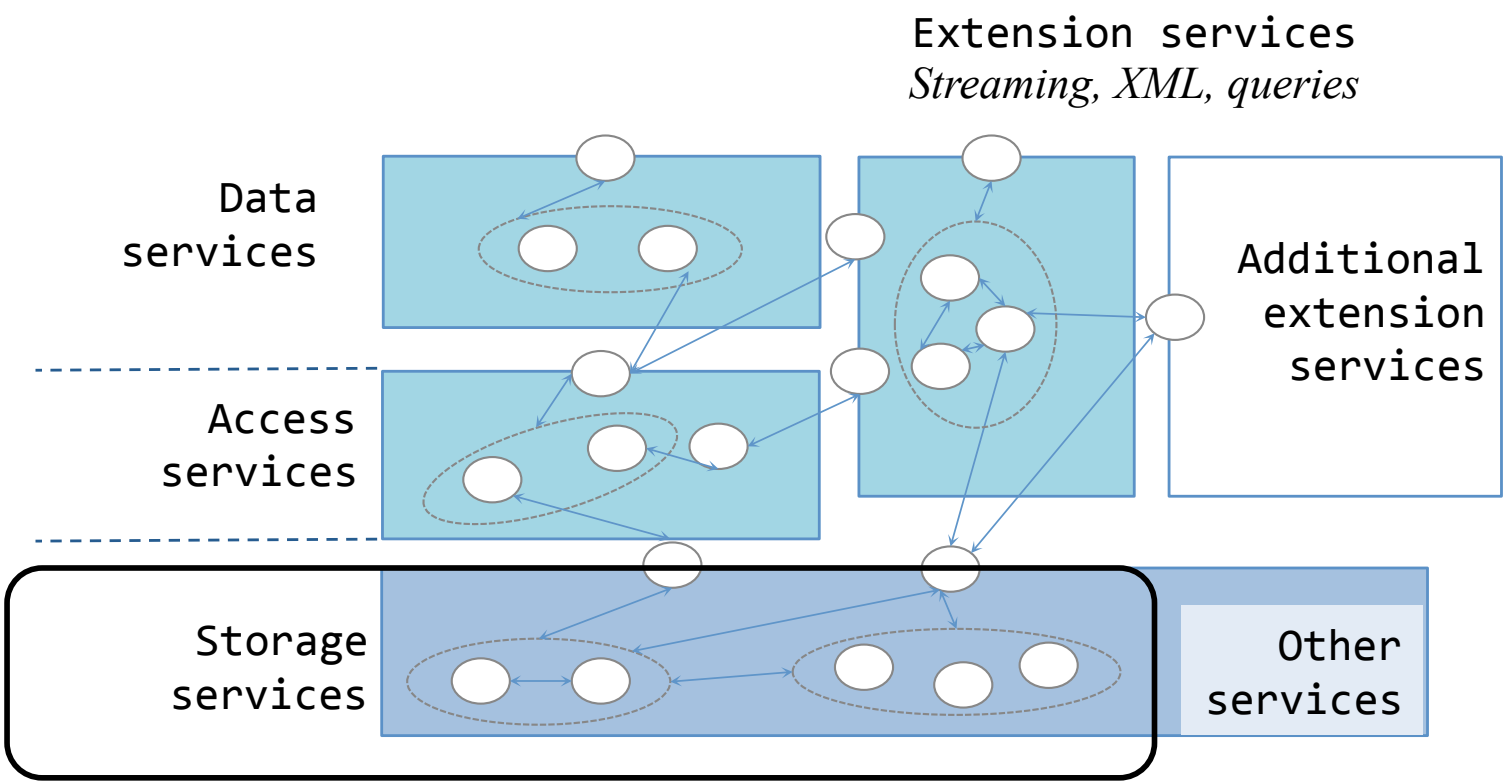
Use the right tool for a given job…



Lack of standardization of models and data storage technologies

(Katsov-2012)

Polyglot persistence



Quality driven benchmark[1]

CHARACTERISTIC	SUBCHARACTERISTIC	METRIC
Reliability	Maturity	API changes
	Availability	Downtime [3]
	Fault tolerance	Node down throughput [3]
	Recoverability	Time to stabilize on node up [3]
Performance and efficiency	Time behaviour	Throughput, latency [2]
	Resource utilisation	CPU, Memory and disk usage [4]

Data stores designed to scale simply

OLTP-style application loads

Read/Write operations
by **thousands/millions** of users

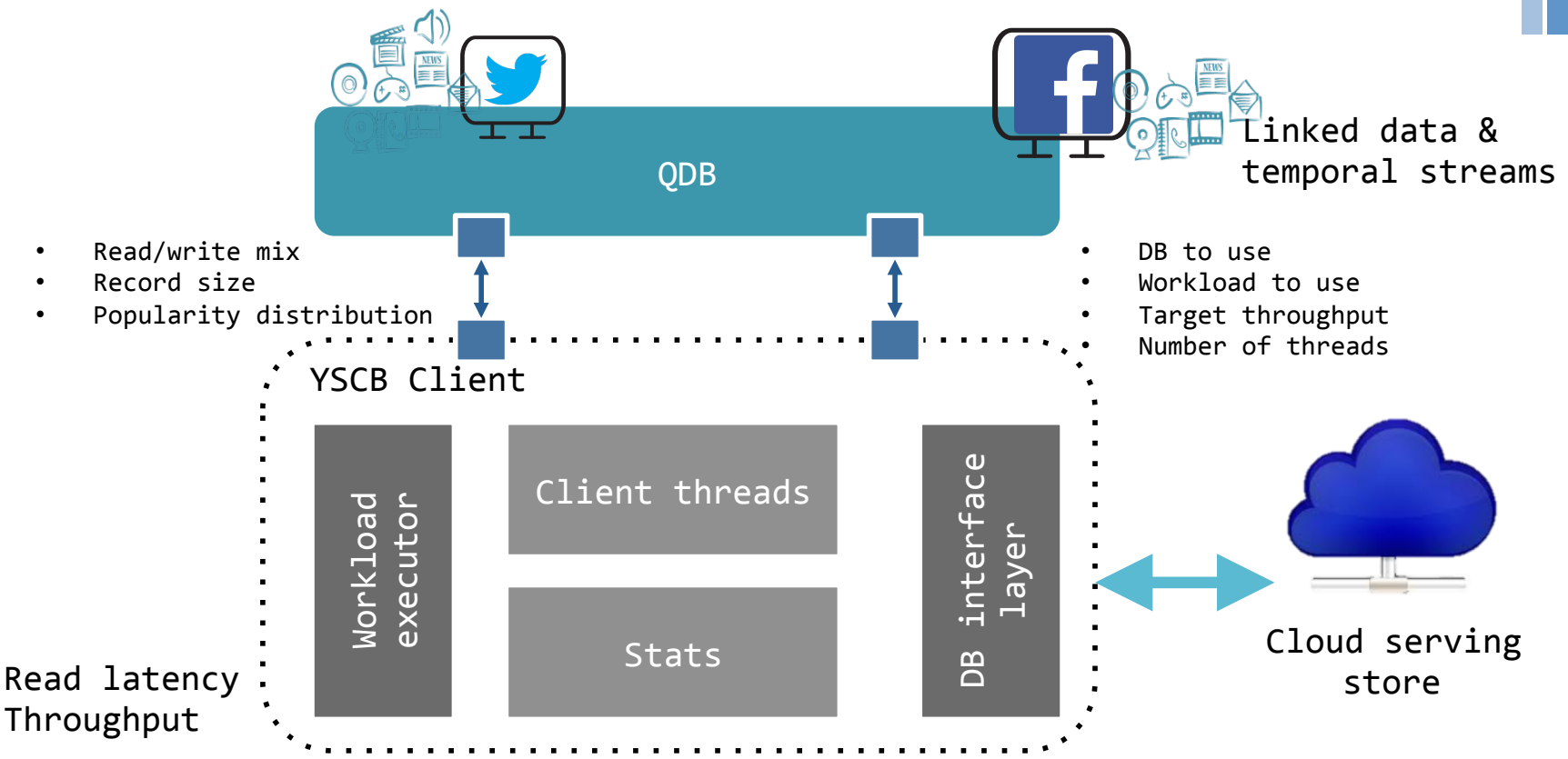
[1]Yahoo Cloud Serving Benchmark, https://github.com/brianfrankcooper/YCSB/wiki

[2] Cooper,B.F.,Silberstein,A.,Tam,E.,Ramakrishnan,R.,Sears,R.:Benchmarking cloud serving systems with YCSB. In: Proceedings of the 1st ACM symposium on Cloud computing. pp. 143–154. SoCC '10, ACM, New York, NY, USA (2010)

[3] Nelubin, D., Engber, B.: Failover Characteristics of leading NoSQL databases. Tech. rep., Thumbtack Technology (2013)

[4] Massie, M.L., Chun, B.N., Culler, D.E.: The Ganglia Distributed Monitoring Sys- tem: Design, Implementation, and Experience. Parallel Computing 30(7) (Jul 2004)

Quality driven benchmark



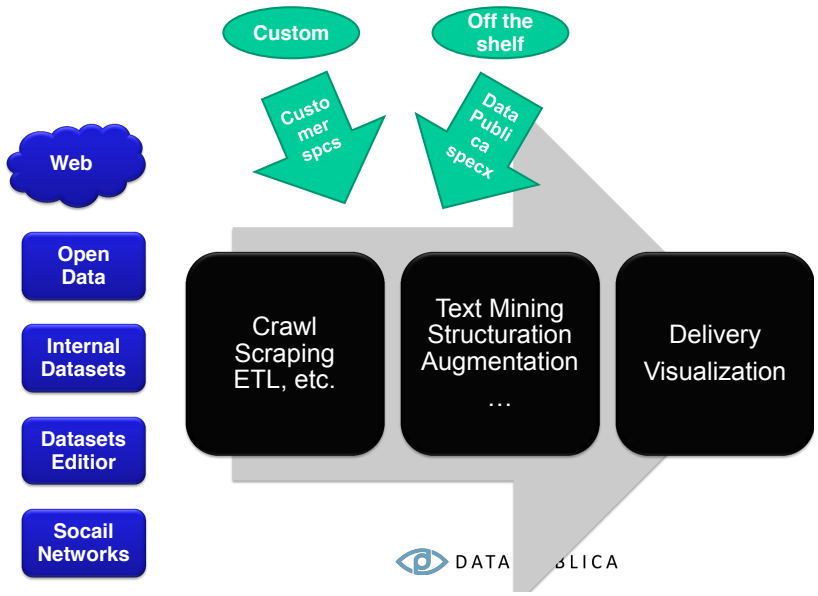
Ongoing work

- **QDB benchmark** extends YCSB: *FaultTolerance, Recoverability and TimeBehaviour*
  - Pivot data model for representing NoSQL stores data models
  - Sample application: Shopping system[1] (`ProductInfo`)
  - Document data stores: MongoDB, Couchbase, VoltDB, Redis, Neo4J
    - Cluster of four Ubuntu 12.04 servers deployed with extra large VM instances (8 virtual cores and 14 GB of RAM) in Windows Azure[2]
- Distributed **polyglot (big) database** engineering
  - Model2Roo: engineering data storage solutions for given data collections
  - ExSchema for supporting the maintenance of a polyglot storage solution

[1] McMurtry, D., Oakley, A., Sharp, J., Subramanian, M., Zhang, H.: Data Access for Highly-Scalable Solutions: Using SQL, NoSQL, and Polyglot Persistence Microsoft patterns & practices, Microsoft (2013)
[2] http://www.windowsazure.com/
[3] http://forge.puppetlabs.com/puppetlabs/
[4]Yahoo Cloud Serving Benchmark, https://github.com/brianfrankcooper/YCSB/wiki

Challenge: open data

- Data journalism
  - http://datauy.org/que-es-una-hackaton/
- Open data services:
  - http://www.data-publica.com
- Open data repositories
  - http://www.infotecarios.com/hackathon-repositorios-de-datos-abiertos-open-data-segunda-parte/
- Open knowledge
  - https://okfn.org
- Datos en Uruguay
  - http://datauy.org/asi-fue-el-dia-mundial-de-los-datos-abiertos-2013/



Challenge: polyglot meets XPeranto

Given a data collection coming from different social networks stored on NoSQL systems (Neo4J and Mongo) *[possibly according to a strategy combining sharding and replication techniques]*, extend the UnQL pivot query language considering

- Data processing operators adapted to query different data models (graph, document). Example query Neo + Mongo and what about Join, Union …
- Assuming concurrent CRUD operations to the stores can you expect query results to be consistent ? How can you tag your results or implement a sharding strategy in order to determine whether results are consistent?
- Querying data represented on different models: How can you exploit the structure of the different stores for expressing queries ? Provide adapted operators? Give generic operators and then rewrite queries?
- Normally, Polyglot solutions tend to solve some data processing issues in the application code. This can be penalizing. Discuss the challenges to address for ensuring that your queries will be able to scale as the collection grows.

Challenge: expected results

- Give the principle of your proposal through a **partial programming solution**, of the operators of your UnQL extension, detail the query evaluation process if U want your solution to scale
  - We ask U to sketch the solution on the polyglot database that we provide consisting of mongo, Neo4J stores
  - https://github.com/jccastrejon/edbt-unql
  - Technical requirements: VMware player 5

When is polyglot persistence pertinent?

- Application essentially composing and serving web pages
  - They only looked up page elements by ID, they had **different needs or availability**, **concurrency** and **no need to share** all their data
  - A problem like this is much better suited to a NoSQL store than the corporate relational DBMS
- **Scaling** to lots of traffic gets harder and harder to do with **vertical scaling**
  - Many NoSQL databases are designed to operate over clusters
  - They can **tackle larger volumes of traffic and data** than is realistic with a single server

Conclusions

- Data are growing big and more heterogeneous and they need new adapted ways to be managed thus the NoSQL movement is gaining momentum
- Data heterogeneity implies different management requirements this is where polyglot persistence comes up
  - Consistency – Availability – Fault tolerance theorem: find the balance !
  - Which data store according to its data model?
  - A lot of programming implied …

Open opportunities if you're interested in this topic!

Merci

Thanks

Gracias

Contact: Genoveva Vargas-Solar, CNRS, LIG-LAFMIA
Genoveva.Vargas@imag.fr
http://www.vargas-solar.com/teaching/

Open source polyglot persistence tools

http://code.google.com/p/exschema/

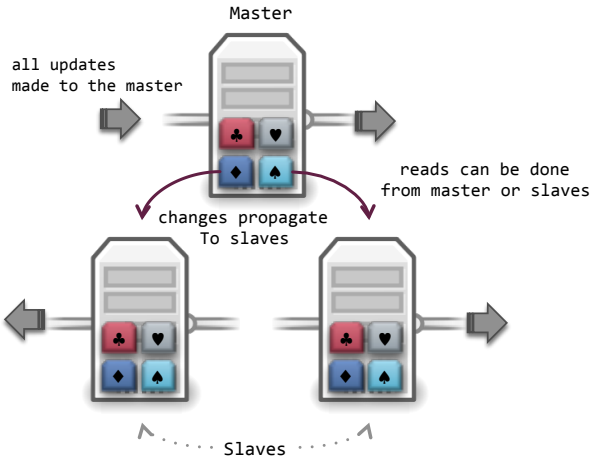
http://code.google.com/p/model2roo/

References

- Eric A., Brewer "Towards robust distributed systems." PODC. 2000
- Rick, Cattell "Scalable SQL and NoSQL data stores." ACM SIGMOD Record 39.4 (2011): 12-27
- Juan Castrejon, Genoveva Vargas-Solar, Christine Collet, and Rafael Lozano, ExSchema: Discovering and Maintaining Schemas from Polyglot Persistence Applications, In Proceedings of the International Conference on Software Maintenance, Demo Paper, IEEE, 2013
- M. Fowler and P. Sadalage. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Pearson Education, Limited, 2012
- C. Richardson, Developing polyglot persistence applications, http://fr.slideshare.net/chris.e.richardson/developing-polyglotpersistenceapplications-gluecon2013

NOSQL STORES: AVAILABILITY AND PERFORMANCE

Replication master - slave

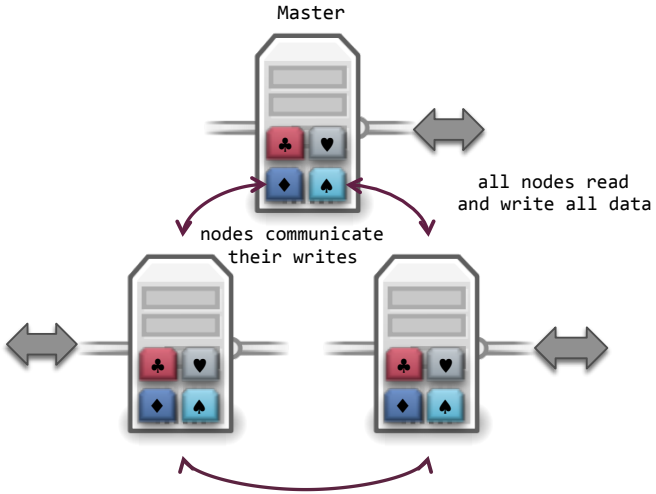


- Makes one node the authoritative copy/replica that handles writes while replica synchronize with the master and may handle reeds
- All replicas have the same weight
  - Replicas can all accept writes
  - The lose of one of them does not prevent access to the data store
- Helps with read scalability but does not help with write scalability
- Read resilience: should the master fail, slaves can still handle read requests
- Master failure eliminates the ability to handle writes until either the master is restored or a new master is appointed
- Biggest complication is consistency
  - Possible write – write conflict
  - Attempt to update the same record at the same time from to different places
- Master is a bottle-neck and a point of failure

Master-slave replication management

- Masters can be appointed
  - Manually when configuring the nodes cluster
  - Automatically: when configuring a nodes cluster one of them elected as master. The master can appoint a new master when the master fails reducing downtime
- Read resilience
  - Read and write paths have to be managed separately to handle failure in the write path and still reads can occur
  - Reads and writes are put in different database connections if the database library accepts it
- Replication comes inevitably with a dark side: **inconsistency**
  - Different clients reading different slaves will see different values if changes have not been propagated to all slaves
  - In the worst case a client cannot read a write it just made
  - Even if master-slave is used for hot backups, if the master fails any updates on to the backup are lost

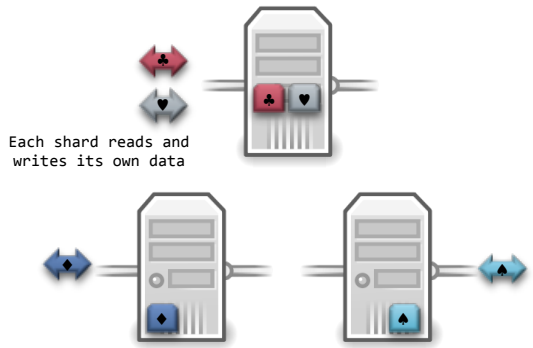
Replication: peer-To-Peer



- Allows writes to any node; the nodes coordinate to synchronize their copies
- The replicas have equal weight

- Deals with inconsistencies
  - Replicas coordinate to avoid conflict
  - Network traffic cost for coordinating writes
  - Unnecessary to make all replicas agree to write, only the majority
  - Survival to the loss of the minority of replicas nodes
  - Policy to merge inconsistent writes
  - Full performance on writing to any replica

Sharding



- Ability to distribute both data and load of simple operations over many servers, with no RAM or disk shared among servers
- A way to horizontally scale **writes**
- Improve **read** performance
- Application/data store support

- Puts different data on separate nodes
  - Each user only talks to one servicer so she gets rapid responses
  - The load should be balanced out nicely between servers
- Ensure that
  - data that is accessed together is clumped together on the same node
  - that clumps are arranged on the nodes to provide best data access

Sharding

Database laws

- Small databases are fast
- Big databases are slow
- Keep databases small

Principle

- Start with a big monolithic database
  - Break into smaller databases
  - Across many clusters
  - Using a key value

Instead of having one million customers information on a single big machine ….

*100 000 customers on **smaller** and **different** machines*

Sharding criteria

- Partitioning
  - Relational: handled by the DBMS (homogeneous DBMS)
  - NoSQL: based on ranging of the k-value
- Federation
  - Relational
    - Combine tables stored in different physical databases
    - Easier with denormalized data
  - NoSQL:
    - Store together data that are accessed together
    - Aggregates unit of distribution

Sharding

Architecture

- Each application server (AS) is running DBS/client
- Each shard server is running
  - a database server
  - replication agents and query agents for supporting parallel query functionality

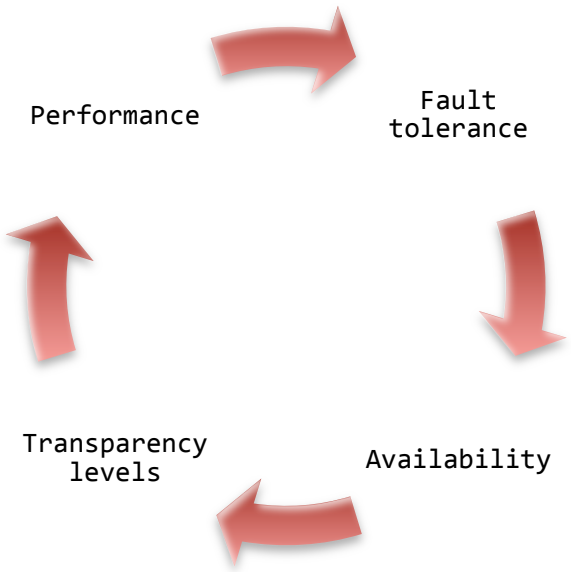
Process

- Pick a dimension that helps sharding easily (customers, countries, addresses)
- Pick strategies that will last a long time as repartition/re-sharding of data is operationally difficult
- This is done according to two different principles
  - **Partitioning**: a partition is a structure that divides a space into tow parts
  - **Federation**: a set of things that together compose a centralized unit but each individually maintains some aspect of autonomy

Customers data is partitioned by ID in shards using an algorithm d to determine which shard a customer ID belongs to

Replication: aspects to consider

- Conditioning



- Important elements to consider
  - Data to duplicate
  - Copies location
  - Duplication model (master – slave / P2P)
  - Consistency model (global – copies)

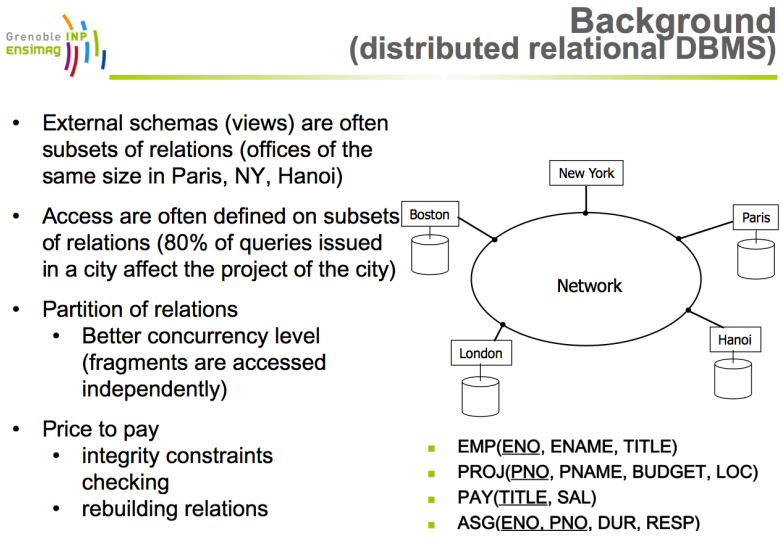
→ **Find a compromise !**

PARTITIONING

A PARTITION IS A STRUCTURE THAT DIVIDES A SPACE INTO TOW PARTS

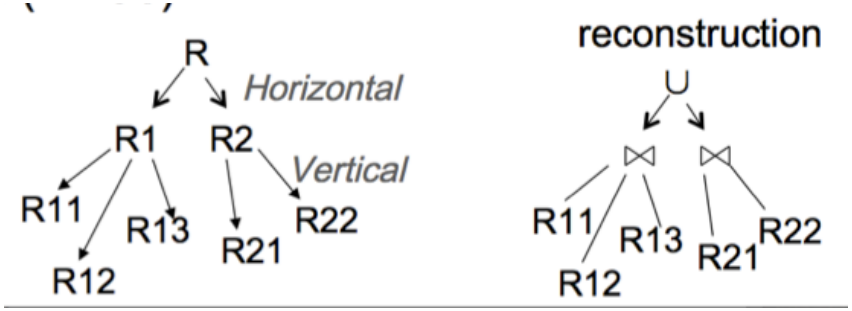
Background: distributed relational databases

- External schemas (views) are often subsets of relations (contacts in Europe and America)
- Access defined on subsets of relations: 80% of the queries issued in a region have to do with contacts of that region
- Relations partition
  - Better concurrency level
  - Fragments accessed independently
- Implications
  - Check integrity constraints
  - Rebuild relations



Fragmentation

- Horizontal
  - Groups of tuples of the same relation
  - Budget < 300 000 or >= 150 000
  - Not disjoint are more difficult to manage
- Vertical
  - Groups attributes of the same relation
  - Separate budget from loc and pname of the relation project
- Hybrid



Fragmentation: rules

Vertical

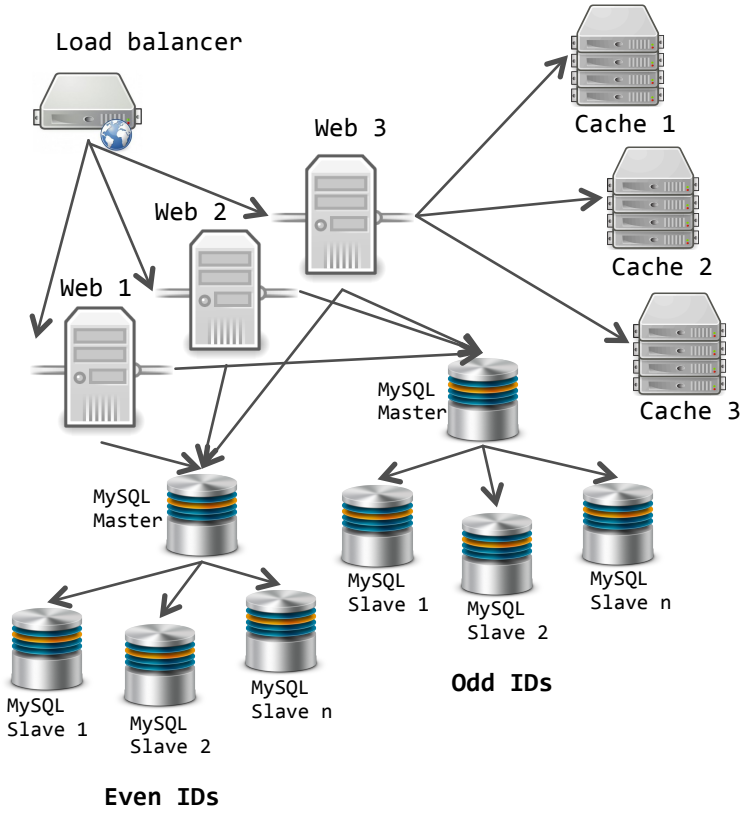
- Clustering
  - Grouping elementary fragments
  - Budget and location information in two relations
- Splitting
  - Decomposing a relation according to affinity relationships among attributes

Horizontal

- Tuples of the same fragment must be statistically homogeneous
  - If t1 and t2 are tuples of the same fragment then t1 and t2 have the same probability of being selected by a query
- Keep important conditions
  - Complete
    - Every tuple (attribute) belongs to a fragment (without information loss)
    - If tuples where budget >= 150 000 are more likely to be selected then it is a good candidate
  - Minimum
    - If no application distinguishes between budget >= 150 000 and budget < 150 000 then these conditions are unnecessary

Sharding: horizontal partitioning

- The entities of a database are split into two or more sets (by row)
- In relational: same schema several physical bases/servers
  - Partition contacts in Europe and America shards where they zip code indicates where the will be found
  - Efficient if there exists some robust and implicit way to identify in which partition to find a particular entity
- Last resort shard
  - Needs to find a sharding function: modulo, round robin, hash – partition, range - partition

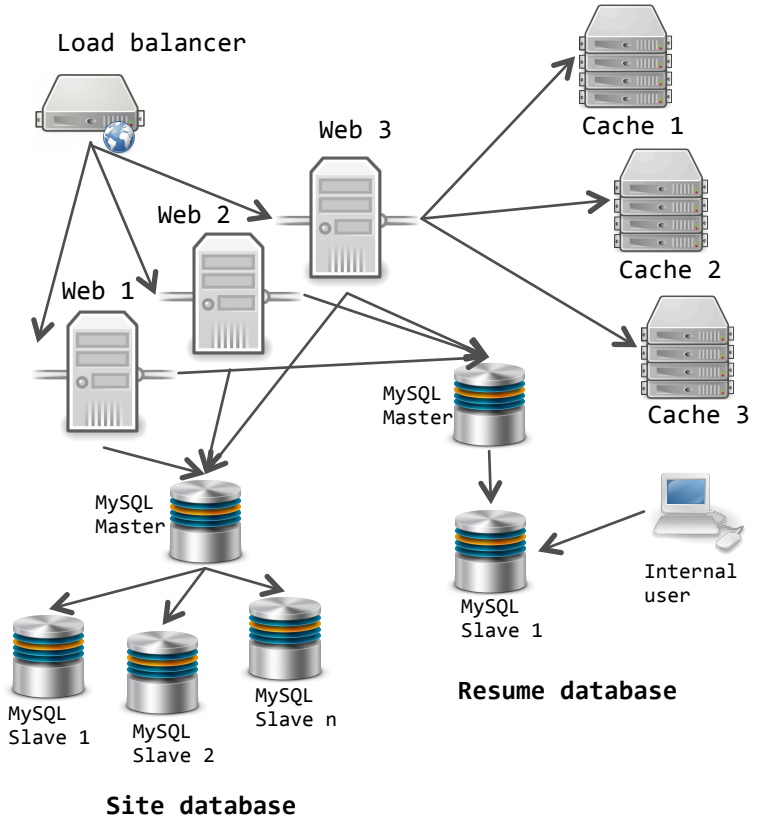


FEDERATION

A FEDERATION IS A SET OF THINGS THAT TOGETHER COMPOSE A CENTRALIZED UNIT BUT EACH INDIVIDUALLY MAINTAINS SOME ASPECT OF AUTONOMY

FEDERATION: vertical SHARDING

- Principle
  - Partition data according to their logical affiliation
  - Put together data that are commonly accessed
- The search load for the large partitioned entity can be split across multiple servers (logical and physical) and not only according to multiple indexes in the same logical server
- Different schemas, systems, and physical bases/ servers
- Shards the components of a site and not only data



NOSQL STORES: PERSISTENCY MANAGEMENT

«*memcached*»

- «*memcached*» is a memory management protocol based on a cache:
  - Uses the key-value notion
  - Information is completly stored in RAM
- «*memcached*» protocol for:
  - Creating, retrieving, updating, and deleting information from the database
  - Applications with their own «*memcached*» manager (Google, Facebook, YouTube, FarmVille, Twitter, Wikipedia)

Storage on disc (1)

- For efficiency reasons, information is stored using the RAM:
  - Work information is in RAM in order to answer to low latency requests
  - Yet, this is not always possible and desirable

➢ **The process of moving data from RAM to disc is called "*eviction*"; this process is configured automatically for every bucket**

Storage on disc (2)

- NoSQL servers support the storage of key-value pairs on disc:
  - **Persistency**–can be executed by loading data, closing and reinitializing it without having to load data from another source
  - **Hot backups**– loaded data are sotred on disc so that it can be reinitialized in case of failures
  - **Storage on disc**– the disc is used when the quantity of data is higher thant the physical size of the RAM, frequently used information is maintained in RAM and the rest es stored on disc

Storage on disc (3)

- Strategies for ensuring:
  - Each node maintains in RAM information on the key-value pairs it stores. Keys:
    - may not be found, or
    - they can be stored in memory or on disc
  - The process of moving information from RAM to disc is asynchronous:
    - The server can continue processing new requests
    - A queue manages requests to disc

- **In periods with a lot of writing requests, clients can be notified that the server is termporaly out of memory until information is evicted**

NOSQL STORES: CONCURRENCY CONTROL

Multi version concurrency control (MVCC)

- ***Objective***: Provide concurrent access to the database and in programming languages to implement transactional memory
- ***Problem***: If someone is reading from a database at the same time as someone else is writing to it, the reader could see a half-written or inconsistent piece of data.
- Lock: readers wait until the writer is done
- MVCC:
  - Each user connected to the database sees a snapshot of the database at a particular instant in time
  - Any changes made by a writer will not be seen by other users until the changes have been completed (until the transaction has been committed
  - When an MVCC database needs to update an item of data it marks the old data as obsolete and adds the newer version elsewhere → multiple versions stored, but only one is the latest
  - Writes can be isolated by virtue of the old versions being maintained
  - Requires (generally) the system to periodically sweep through and delete the old, obsolete data objects