CompSci 516
Database Systems

Lecture 20
NoSQL
and
Column Store

Instructor: Sudeepa Roy

Reading Material

NOSQL:

- "Scalable SQL and NoSQL Data Stores"

Rick Cattell, SIGMOD Record, December 2010 (Vol. 39, No. 4)

- see webpage http://cattell.net/datastores/ for updates and more pointers
- MongoDB manual: https://docs.mongodb.com/manual/

Column Store:

- D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos and S. Madden. *The Design and Implementation of Modern Column-Oriented Database Systems*. Foundations and Trends in Databases, vol. 5, no. 3, pp. 197–280, 2012.
- See VLDB 2009 tutorial: http://nms.csail.mit.edu/~stavros/pubs/tutorial2009-column_stores.pdf

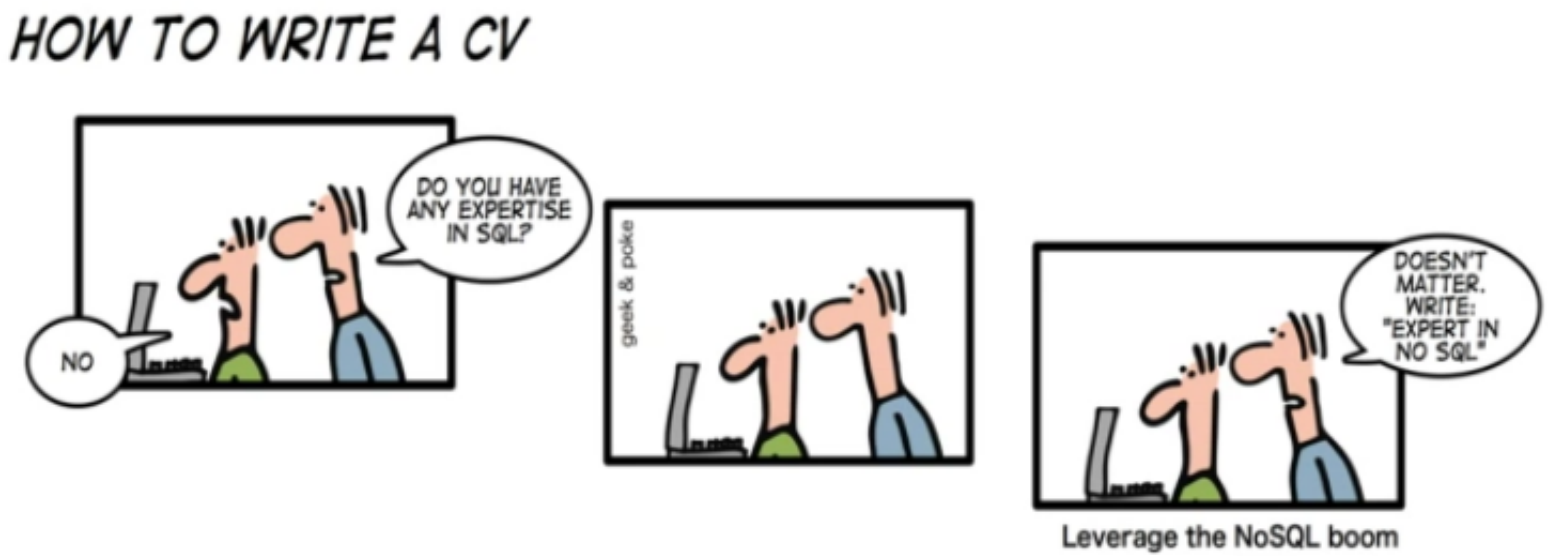
Optional:

- "Dynamo: Amazon's Highly Available Key-value Store" By Giuseppe DeCandia et. al. SOSP 2007

- "Bigtable: A Distributed Storage System for Structured Data" Fay Chang et. al. OSDI 2006

NoSQL

See the optional/additional slides on MongoDB
on the course website
May be useful for HW3


HOW TO WRITE A CV
DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"
Leverage the NoSQL boom

So far -- RDBMS

- Relational Data Model
- Relational Database Systems (RDBMS)
- RDBMSs have
  - a complete pre-defined fixed schema
  - a SQL interface
  - and ACID transactions

Today

- NoSQL: ”new” database systems
  - not typically RDBMS
  - relax on some requirements, gain efficiency and scalability
- New systems choose to use/not use several concepts we learnt so far
  - e.g. “System ---” does not use locks but uses multi-version CC (MVCC) or,
  - “System ---” uses asynchronous replication
- therefore, it is important to understand the basics (Lectures 1-18) even if they are not used in some new systems!

Warnings!

- Material from Cattell's paper (2010-11) – some info will be outdated
  - see webpage http://cattell.net/datastores/ for updates and more pointers
- We will focus on the basic ideas of NoSQL systems
- Optional reading slides at the end on MongoDB
  - may be useful for HW3
  - there are also comparison tables in the Cattell's paper if you are interested

OLAP vs. OLTP

- OLTP (OnLine Transaction Processing)
  - Recall transactions!
  - Multiple concurrent read-write requests
  - Commercial applications (banking, online shopping)
  - Data changes frequently
  - ACID properties, concurrency control, recovery
- OLAP (OnLine Analytical Processing)
  - Many aggregate/group-by queries – multidimensional data
  - Data mostly static
  - Will study OLAP Cube soon

New Systems

- We will examine a number of SQL and so- called "NoSQL" systems or "data stores"
- Designed to scale simple OLTP-style application loads
  - to do updates as well as reads
  - in contrast to traditional DBMSs and data warehouses
  - to provide good horizontal scalability (?) for simple read/write database operations distributed over many servers
- Originally motivated by Web 2.0 applications
  - these systems are designed to scale to thousands or millions of users

New Systems vs. RDMS

- When you study a new system, compare it with RDBMS-s on its
  - data model
  - consistency mechanisms
  - storage mechanisms
  - durability guarantees
  - availability
  - query support
- These systems typically sacrifice some of these dimensions
  - e.g. database-wide transaction consistency, in order to achieve others, e.g. higher availability and scalability

NoSQL

- Many of the new systems are referred to as "NoSQL" data stores
- NoSQL stands for "Not Only SQL" or "Not Relational"
  - not entirely agreed upon
- Next: six key features of NoSQL systems

NoSQL: Six Key Features

1. the ability to horizontally scale “simple operations” throughput over many servers
2. the ability to replicate and to distribute (partition) data over many servers
3. a simple call level interface or protocol (in contrast to SQL binding)
4. a weaker concurrency model than the ACID transactions of most relational (SQL) database systems
5. efficient use of distributed indexes and RAM for data storage
6. the ability to dynamically add new attributes to data records

Important Examples of New Systems

- Three systems provided a "proof of concept" and inspired many other data stores

1. Memcached
2. Amazon's Dynamo
3. Google's BigTable

1. Memcached: main features

- popular open source cache
- supports distributed hashing (later)
- demonstrated that in-memory indexes can be highly scalable, distributing and replicating objects over multiple nodes

2. Dynamo : main features

- pioneered the idea of eventual consistency as a way to achieve higher availability and scalability
- data fetched are not guaranteed to be up-to-date
- but updates are guaranteed to be propagated to all nodes eventually

3. BigTable : main features

- demonstrated that persistent record storage could be scaled to thousands of nodes
- “column families”

- https://cloud.google.com/bigtable/
- https://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf

BASE (not ACID ☺)

- Recall ACID for RDBMS desired properties of transactions:
  - Atomicity, Consistency, Isolation, and Durability

- NOSQL systems typically do not provide ACID

- Basically Available
- Soft state
- Eventually consistent

ACID vs. BASE

- The idea is that by giving up ACID constraints, one can achieve much higher performance and scalability

- The systems differ in how much they give up
  - e.g. most of the systems call themselves "eventually consistent", meaning that updates are eventually propagated to all nodes
  - but many of them provide mechanisms for some degree of consistency, such as multi-version concurrency control (MVCC)

"CAP" Theorem

- Often Eric Brewer's CAP theorem cited for NoSQL

- A system can have only two out of three of the following properties:
- Consistency
  - do all clients see the same data?
- Availability
  - is the system always on?
- Partition-tolerance
  - even if communication is unreliable, does the system function?

- The NoSQL systems generally give up consistency
  - However, the trade-offs are complex

Two foci for NoSQL systems

1. "Simple" operations

2. Horizontal Scalability

1. “Simple” Operations

- Reading or writing a small number of related records in each operation
  - e.g. key lookups
  - reads and writes of one record or a small number of records
- This is in contrast to complex queries, joins, or read-mostly access
- Inspired by web, where millions of users may both read and write data in simple operations
  - e.g. search and update multi-server databases of electronic mail, personal profiles, web postings, wikis, customer records, online dating records, classified ads, and many other kinds of data

2. Horizontal Scalability

- Shared-Nothing Horizontal Scaling
- The ability to distribute both the data and the load of these simple operations over many servers
  - with no RAM or disk shared among the servers
- Not "vertical" scaling
  - where a database system utilizes many cores and/or CPUs that share RAM and disks
- Some of the systems we describe provide both vertical and horizontal scalability

2. Horizontal vs. Vertical Scaling

- Effective use of multiple cores (vertical scaling) is important
  - but the number of cores that can share memory is limited
- horizontal scaling generally is less expensive
  - can use commodity servers
- Note: horizontal and vertical partitioning are not related to horizontal and vertical scaling (Lecture 18)
  - except that they are both useful for horizontal scaling

What is different in NOSQL systems

- When you study a new NOSQL system, notice how it differs from RDBMS in terms of

1. Concurrency Control
2. Data Storage Medium
3. Replication
4. Transactions

Choices in NOSQL systems: 1. Concurrency Control

a) Locks
   - some systems provide one-user-at-a-time read or update locks
   - MongoDB provides locking at a field level

b) MVCC

c) None
   - do not provide atomicity
   - multiple users can edit in parallel
   - no guarantee which version you will read

d) ACID
   - pre-analyze transactions to avoid conflicts
   - no deadlocks and no waits on locks

Choices in NOSQL systems: 2. Data Storage Medium

a) Storage in RAM
   - snapshots or replication to disk
   - poor performance when overflows RAM

b) Disk storage
   - caching in RAM

Choices in NOSQL systems: 3. Replication

- whether mirror copies are always in sync

a) Synchronous

b) Asynchronous
   - faster, but updates may be lost in a crash

c) Both
   - local copies synchronously, remote copies asynchronously

Choices in NOSQL systems: 4. Transaction Mechanisms

a) support
b) do not support
c) in between
   - support local transactions only within a single object or "shard"
   - shard = a horizontal partition of data in a database

Comparison from Cattell's paper (2011)

System	Conc Contol	Data Storage	Repli-cation	Tx
Redis	Locks	RAM	Async	N
Scalaris	Locks	RAM	Sync	L
Tokyo	Locks	RAM or disk	Async	L
Voldemort	MVCC	RAM or BDB	Async	N
Riak	MVCC	Plug-in	Async	N
Membrain	Locks	Flash + Disk	Sync	L
Membase	Locks	Disk	Sync	L
Dynamo	MVCC	Plug-in	Async	N
SimpleDB	None	S3	Async	N
MongoDB	Locks	Disk	Async	N
Couch DB	MVCC	Disk	Async	N

Terrastore	Locks	RAM+	Sync	L
HBase	Locks	Hadoop	Async	L
HyperTable	Locks	Files	Sync	L
Cassandra	MVCC	Disk	Async	L
BigTable	Locks+s tamps	GFS	Sync+ Async	L
PNUTs	MVCC	Disk	Async	L
MySQL Cluster	ACID	Disk	Sync	Y
VoltDB	ACID, no lock	RAM	Sync	Y
Clustrix	ACID, no lock	Disk	Sync	Y
ScaleDB	ACID	Disk	Sync	Y
ScaleBase	ACID	Disk	Async	Y
NimbusDB	ACID, no lock	Disk	Sync	Y

Data Model Terminology for NoSQL

- Unlike SQL/RDBMS, the terminology for NoSQL is often inconsistent
  - we are following notations in Cattell's paper
- All systems provide a way to store scalar values
  - e.g. numbers and strings
- Some of them also provide a way to store more complex nested or reference values

Data Model Terminology for NoSQL

- The systems all store sets of attribute-value pairs
  - but use four different data structures

1. Tuple
2. Document
3. Extensible Record
4. Object

1. Tuple

- Same as before
- A "tuple" is a row in a relational table
  - attribute names are pre-defined in a schema
  - the values must be scalar
  - the values are referenced by attribute name
  - in contrast to an array or list, where they are referenced by ordinal position

2. Document

- Allows values to be nested documents or lists as well as scalar values
  - think about XML or JSON
- The attribute names are dynamically defined for each document at runtime
- A document differs from a tuple in that the attributes are not defined in a global schema
  - and a wider range of values are permitted

3. Extensible Record

- A hybrid between a tuple and a document
- families of attributes are defined in a schema
- but new attributes can be added (within an attribute family) on a per-record basis
- Attributes may be list-valued

4. Object

- Analogous to an object in programming languages
  - but without the procedural methods

- Values may be references or nested objects

Example NOSQL systems

- Key-value Stores:
  - Project Voldemort, Riak, Redis, Scalaris, Tokyo Cabinet, Memcached/Membrain/Membase
- Document Stores:
  - Amazon SimpleDB, CouchDB, MongoDB, Terrastore
- Extensible Record Stores:
  - Hbase, HyperTable, Cassandra, Yahoo's PNUTS
- Relational Databases:
  - MySQL Cluster, VoltDB, Clustrix, ScaleDB, ScaleBase, NimbusDB, Google Megastore (a layer on BigTable)

SQL vs. NOSQL

Arguments for both sides
still a controversial topic

Why choose RDBMS over NoSQL : 1/3

1. If new relational systems can do everything that a NoSQL system can, with analogous performance and scalability (?), and with the convenience of transactions and SQL, NoSQL is not needed

2. Relational DBMSs have taken and retained majority market share over other competitors in the past 30 years
   - (network, object, and XML DBMSs)

Why choose RDBMS over NoSQL : 2/3

3. Successful relational DBMSs have been built to handle other specific application loads in the past:
   - read-only or read-mostly data warehousing
   - OLTP on multi-core multi-disk CPUs
   - in-memory databases
   - distributed databases, and
   - now horizontally scaled databases

Why choose RDBMS over NoSQL : 3/3

4. While no "one size fits all" in the SQL products themselves, there is a common interface with SQL, transactions, and relational schema that give advantages in training, continuity, and data interchange

Why choose NoSQL over RDBMS : 1/3

1. We haven't yet seen good benchmarks showing that RDBMSs can achieve scaling comparable with NoSQL systems like Google's BigTable

2. If you only require a lookup of objects based on a single key
   - then a key-value store is adequate and probably easier to understand than a relational DBMS
   - Likewise for a document store on a simple application: you only pay the learning curve for the level of complexity you require

Why choose NoSQL over RDBMS : 2/3

3. Some applications require a flexible schema
   - allowing each object in a collection to have different attributes
   - While some RDBMSs allow efficient "packing" of tuples with missing attributes, and some allow adding new attributes at runtime, this is uncommon

Why choose NoSQL over RDBMS : 3/3

4. A relational DBMS makes "expensive" (multi- node multi-table) operations "too easy"
   - NoSQL systems make them impossible or obviously expensive for programmers

5. While RDBMSs have maintained majority market share over the years, other products have established smaller but non-trivial markets in areas where there is a need for particular capabilities
   - e.g. indexed objects with products like BerkeleyDB, or graph-following operations with object-oriented DBMSs

Column Store

Row vs. Column Store

- Row store
  - store all attributes of a tuple together
  - storage like “row-major order” in a matrix
- Column store
  - store all rows for an attribute (column) together
  - storage like “column-major order” in a matrix
- e.g.
  - MonetDB, Vertica (earlier, C-store), SAP/Sybase IQ, Google Bigtable (with column groups)

What is a column-store?

+ easy to add/modify a record

- might read in unnecessary data

+ only need to read in relevant data

- tuple writes require multiple accesses

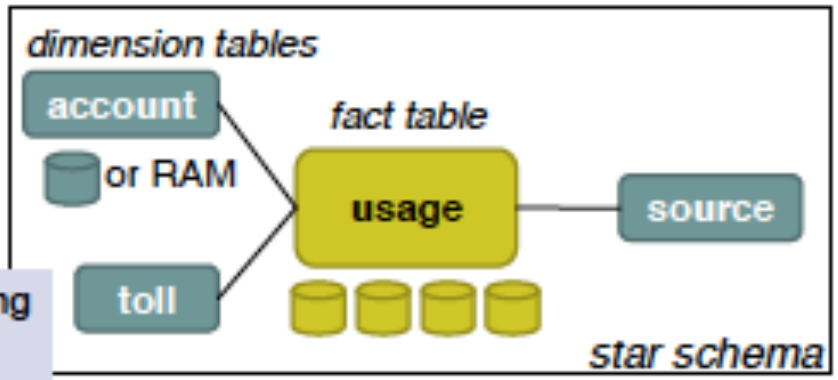
=> suitable for read-mostly, read-intensive, large data repositories

Ack: Slide from VLDB 2009 tutorial on Column store

Telco Data Warehousing example

1. Typical DW installation
1. Real-world example

"One Size Fits All? - Part 2: Benchmarking Results" Stonebraker et al. CIDR 2007




```
QUERY 2
SELECT account.account_number,
sum (usage.toll_airtime),
sum (usage.toll_price)
FROM usage, toll, source, account
WHERE usage.toll_id = toll.toll_id
AND usage.source_id = source.source_id
AND usage.account_id = account.account_id
AND toll.type_ind in ('AE'. 'AA')
AND usage.toll_price > 0
AND source.type != 'CIBER'
AND toll.rating_method = 'IS'
AND usage.invoice_date = 20051013
GROUP BY account.account_number
```

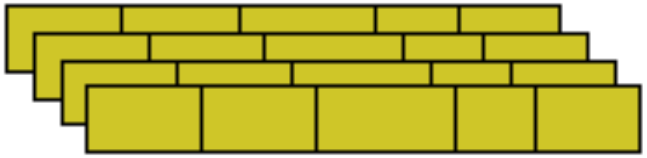
	Column-store	*Row-store*
Query 1	*2.06*	*300*
Query 2	*2.20*	*300*
Query 3	*0.09*	*300*
Query 4	*5.24*	*300*
Query 5	*2.88*	*300*

Why? Three main factors (next slides)

Ack: Slide from VLDB 2009 tutorial on Column store

Telco example explained (1/3): *read efficiency*

row store



read pages containing entire rows

one row = 212 columns!

is this typical? (it depends)

What about vertical partitioning? (it does not work with ad-hoc queries)

column store

read only columns needed

in this example: 7 columns

caveats:

- "select * " not any faster
- clever disk prefetching
- clever tuple reconstruction

Ack: Slide from VLDB 2009 tutorial on Column store

Telco example explained (2/3): *compression efficiency*

- Columns compress better than rows
  - Typical row-store compression ratio 1 : 3
  - Column-store 1 : 10

- Why?
  - Rows contain values from different domains
    => more entropy, difficult to dense-pack
  - Columns exhibit significantly less entropy
  - Examples:

Male, Female, Female, Female, Male
1998, 1998, 1999, 1999, 1999, 2000

  - Caveat: CPU cost (use lightweight compression)

Ack: Slide from VLDB 2009 tutorial on Column store

Telco example explained (3/3): *sorting & indexing efficiency*

- Compression and dense-packing free up space
  - Use multiple overlapping column collections
  - Sorted columns compress better
  - Range queries are faster
  - Use sparse clustered indexes

Ack: Slide from VLDB 2009 tutorial on Column store