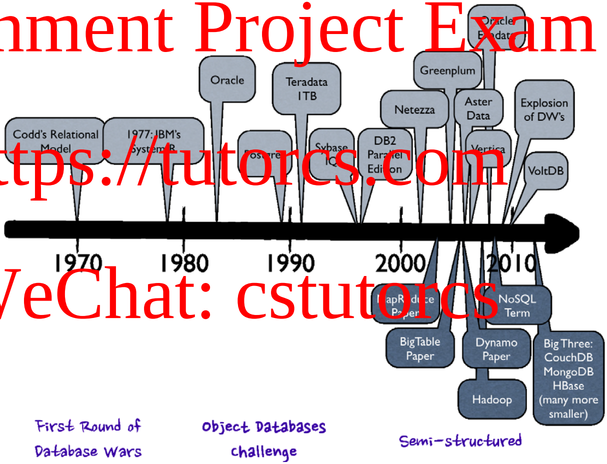
NoSQL Databases – Part 1

Introduction

A Historical View on Database Development[1]



1 http://www.benstopford.com

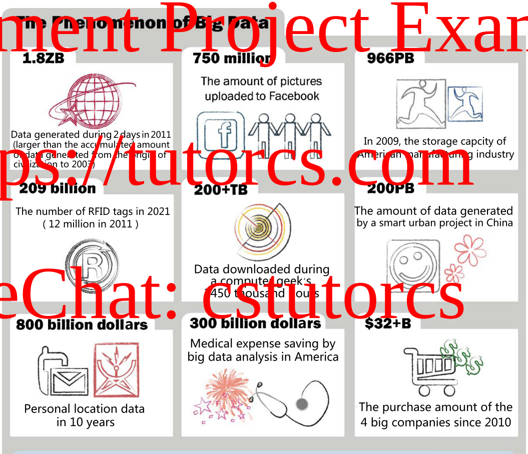
NoSQL - Not only SQL

- A broad class of non-relational databases that do not use SQL as their query language.

- Pioneered by Web 2.0 companies with *huge, growing data* and *infrastructure needs*, e.g.,

  - Amazon introduced *Dynamo*
  - Google developed *Bigtable*

The Need of NoSQL Databases - Big Data[1]

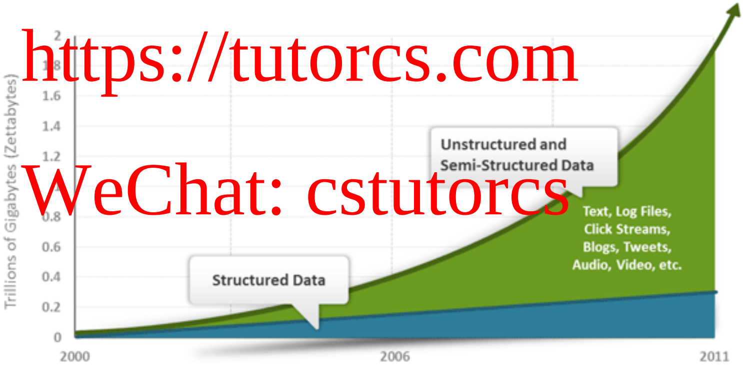


[1] Big Data: A Survey, M Chen, S. Mao, and Y. Liu, Mobile Networks and Applications, 19(2), pages 171–209, 2014

Australian National University

The Need of NoSQL Databases - Big Data

- **Scale of Big Data**: terabytes, petabytes, exabytes, zettabytes, . . .
- **Nature of Big Data**: 3Vs (volume, velocity and variety)



Source: IDC 2011 Digital universe study

DB-Engines Ranking (http://db-engines.com/en/ranking)

312 systems in ranking, December 2016

Rank			DBMS	Database Model	Score		
Dec 2016	Nov 2016	Dec 2015			Dec 2016	Nov 2016	Dec 2015
1.	1.	1.	Oracle	Relational DBMS	1404.40	-8.60	-93.15
2.	2.	2.	MySQL	Relational DBMS	1374.41	+0.85	+75.87
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1226.66	+12.86	+103.50
4.	4.	↑ 5.	PostgreSQL	Relational DBMS	330.02	+4.20	+49.92
5.	5.	↓ 4.	MongoDB	Document store	328.68	+3.21	+27.29
6.	6.	6.	DB2	Relational DBMS	184.34	+2.89	-11.78
7.	7.	↑ 8.	Cassandra	Wide column store	134.28	+0.31	+3.44
8.	8.	↓ 7.	Microsoft Access	Relational DBMS	124.70	-1.27	-15.51
9.	9.	↑ 10.	Redis	Key-value store	119.89	+4.35	+19.36
10.	10.	↓ 9.	SQLite	Relational DBMS	110.83	-1.17	+9.98
11.	11.	↑ 13.	Elasticsearch	Search engine	103.27	+0.70	+26.71
12.	12.	↑ 14.	Teradata	Relational DBMS	73.37	-1.79	-2.34
13.	13.	↓ 11.	SAP Adaptive Server	Relational DBMS	70.42	+0.26	-11.05
14.	14.	↓ 12.	Solr	Search engine	69.00	+0.64	-10.15
15.	15.	↑ 16.	HBase	Wide column store	58.63	-0.11	+4.38
16.	16.	↑ 18.	Splunk	Search engine	54.92	+0.19	+11.06
17.	17.	17.	FileMaker	Relational DBMS	54.12	+0.20	+4.00
18.	18.	↑ 19.	SAP HANA	Relational DBMS	51.77	+2.50	+12.91
19.	19.	↓ 15.	Hive	Relational DBMS	49.40	+0.28	-5.87
20.	20.	↑ 23.	MariaDB	Relational DBMS	44.09	+1.42	+16.35
21.	21.	21.	Neo4j	Graph DBMS	36.83	+0.08	+3.64

A Battle between SQL and NoSQL

Why Relational Databases?

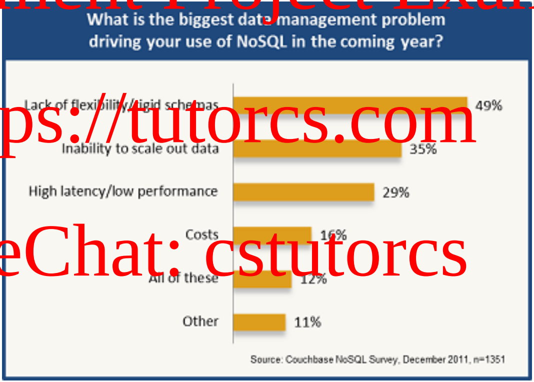
- **Relational databases have ruled the database world for several decades ...**
  - *Simple concepts*, i.e., a database contains tables (called relations), and each table is made up of columns and rows.
  - A logical data model with physical *data independence*
  - A clear separation between *schema and instance*
  - A solid *mathematical foundation*, i.e., set theory, first-order logic, algebra, etc.
  - The standard query and manipulation language - *SQL*
  - *Transactions with ACID properties* (Atomicity, Consistency, Isolation, Durability)

Why Not Relational Databases?

- **Sometimes, relational databases are not the best solution ...**

  - Are *relations* (and their *schemas*) too rigid?
  - Does SQL become tedious and error-prone when handling complex queries?
  - Can we eliminate *joins* so as to improve performance?
  - Is ACID necessary?
  - At *what scale* is a database used (terabyte, petabyte or exabyte)?
  - Do you just need *a very small subset of features* that the relational DBMSs have?

Reasons for Moving to NoSQL



1. Flexibility of Schemas

- Data modeling is as *important* for NoSQL databases as it is for relational databases. But...the modelling approaches are *quite different*:
  - Relational modeling is driven by the structure of available data.

    e.g., **What kind of information do we have?**
  - NoSQL modeling is driven by application-specific operation patterns.

    e.g., **What kinds of questions do we have?**

- Relational databases are not good for *managing hierarchical or graph-like data*, but many of NoSQL databases are.
- Relational databases require *pre-defined schemas* but NoSQL databases have *no fixed schemas*.

2. Scalability

- Many NoSQL databases are driven by **the need to scale**.
- **Shared-nothing** (SN) vs. **shared-everything** (SE): whether to share disk and memory between nodes.

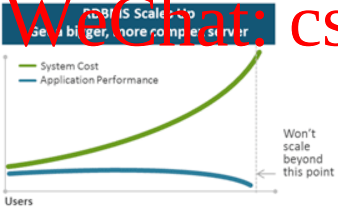
  - **SN:** may have near-linear and unlimited scalability but with design challenges, e.g., Google's Bigtable.
  - **SE:** has no "data shipping" issue but is limited by shared resources, e.g. IBM DB2.

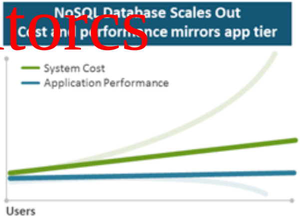
- **Scale up** vs. **scale out**
  - **Scale up** (vertically): add resources to a single node in a system, e.g. CPUs or memory.
  - **Scale out** (horizontally): add more nodes to a system, e.g. web servers.

2. Scalability

- **Relational databases**
  - Can **scale up** by getting faster hardware, but cannot easily **scale out** at an acceptable cost and beyond certain point under ACID constraints.

- **NoSQL databases**
  - Often designed to **scale out** by leveraging commodity hardware and free software, providing an inexpensive solution for scalability.





3. Performance

- Relational databases were invented in a way that **implementation techniques are abstracted away from the user**.

- NoSQL databases promote **exposing the implementation techniques to the programmer**

  Question: *NoSQL databases just need programmers, not DBAs?*

- *Query performance* is often one of the strengths of NoSQL databases, particularly when handling complex-valued data (because they de-normalise data and don't use join).

4. Costs

- NoSQL solutions are often less expensive than RDBMSs, especially when dealing with large-scale data sets.

- A **scale out approach** is usually cheaper than the **scale up alternative**.
  - Many NoSQL databases are **open source**, while **licensing costs** of commercial RDBMSs can be quite expensive.

  - NoSQL databases often leverage **commodity servers** to scale out, while RDBMs tends to rely on expensive **proprietary servers and storage systems**

CAP Theorem

- **CAP Theorem** was proposed by Eric Brewer (UC Berkeley)[1] and proven by Gilbert and Lynch (MIT)[2].

  - **Consistency**

    All users see the same data at the same time.

  - **Availability**

    All users can always read and write data.

  - **Partition tolerance**

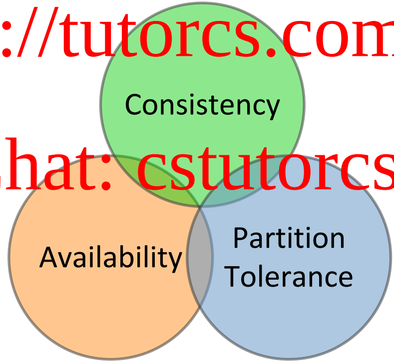
    The system works well with network partitions.

[1] E. Brewer, Towards robust distributed systems, PODC, 2000.

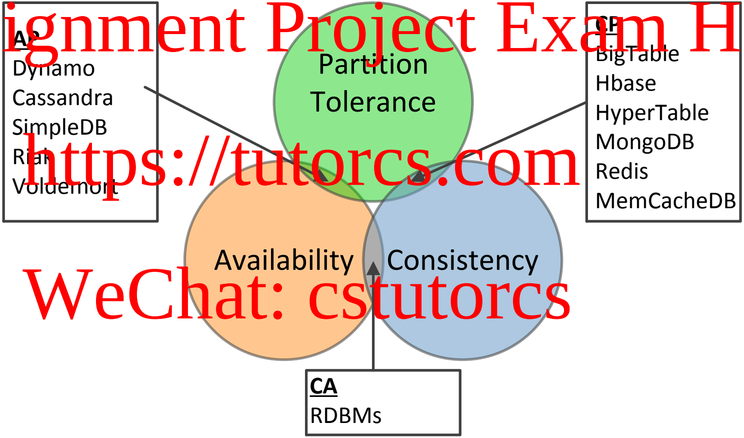
[2] S. Gilbert and N. Lynch, Brewer's conjecture and the feasibility of consistent, available, partition-tolerant Web services. ACM SIGACT News, 2002

CAP Theorem

- CAP Theorem comes to life **as an application scales** (i.e., distributed data management systems).
- A distributed data management system can only have **two out of these three** properties.



CAP Theorem[3]



[3] CAP Twelve Years Later: How the “Rules” Have Changed, Eric Brewer, https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed

ACID

- RDBMSs support the ACID properties for database transactions.

  - **Atomicity**: the execution of each transaction as atomic, i.e., **either all operations are completed or not done at all**.
  - **Consistency**: before and after each transaction, database will be in a consistent state.
  - **Isolation**: execution results of each transaction should be **unaffected by other concurrently executing transactions.**
  - **Durability**: once the DBMS informs the user that a transaction has been successfully completed, **its effects should persist in the database**.

Question: *What kinds of applications ACID properties will be useful for?*

BASE

- NoSQL often uses a model weaker than ACID, called BASE.

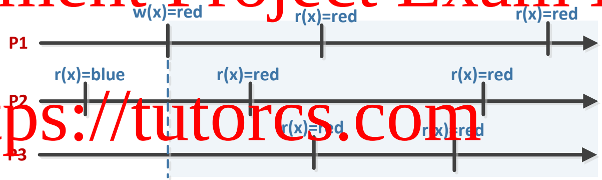
  - **Basically available:** The system may have partial failures. If a single node fails, part of the data won't be available, but the entire data layer stays operational.
  - **Soft state:** The state of the system could change over time (even during times without input), because there may be changes going on due to "eventual consistency".

  - **Eventual consistency:** Given a sufficiently long period of time, all updates can be expected to propagate eventually through the system and the replicas will be consistent.

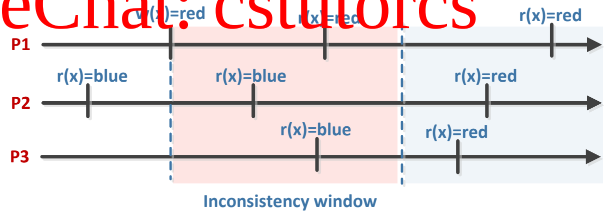
Question: *What kinds of applications will BASE be useful for?*

Consistency Models

- Strong consistency



- Eventual consistency



ACID vs BASE

- There is **a continuum between ACID and BASE** in distributed database systems.
- Depending on your problems, you decide how close you want to be to one end of the continuum or the other.

ACID	BASE
Strong consistency	Weak consistency (stale data OK)
Isolation	Approximate answers OK
Focus on "commit"	Best effort
Nested transactions	Simpler! Faster
Availability?	Availability first
Conservative (pessimistic)	Aggressive (optimistic)
Difficult evolution (e. g. schema)	Easier evolution

Influential NoSQL Solutions

- Companies like Google, Facebook, Amazon, LinkedIn, Baidu and Twitter all use NoSQL in one way or another.

Main Categories of NoSQL Solutions

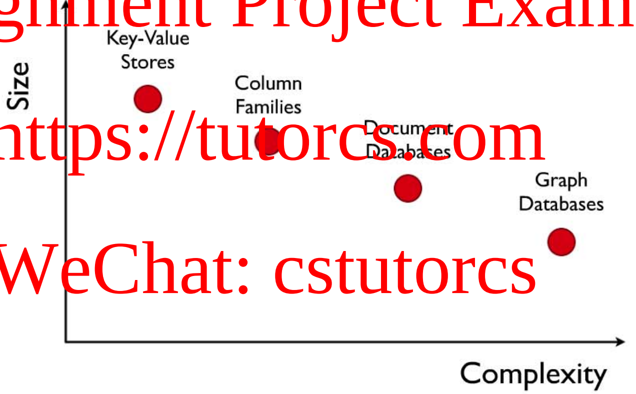
- NoSQL databases are mainly categorized according to their *data models*:

  - **Key-value data stores**
  - **Column-oriented data stores**

  - **Document-oriented data stores**
  - **Graph databases**

Australian National University

NoSQL Data Models[1]



[1] Figure taken from:
http://www.slideshare.net/emileifrem/nosql-east-a-nosql-overview-and-the-benefits-of-graph-databases