CIB Session 9th
Intro to NoSQL Database

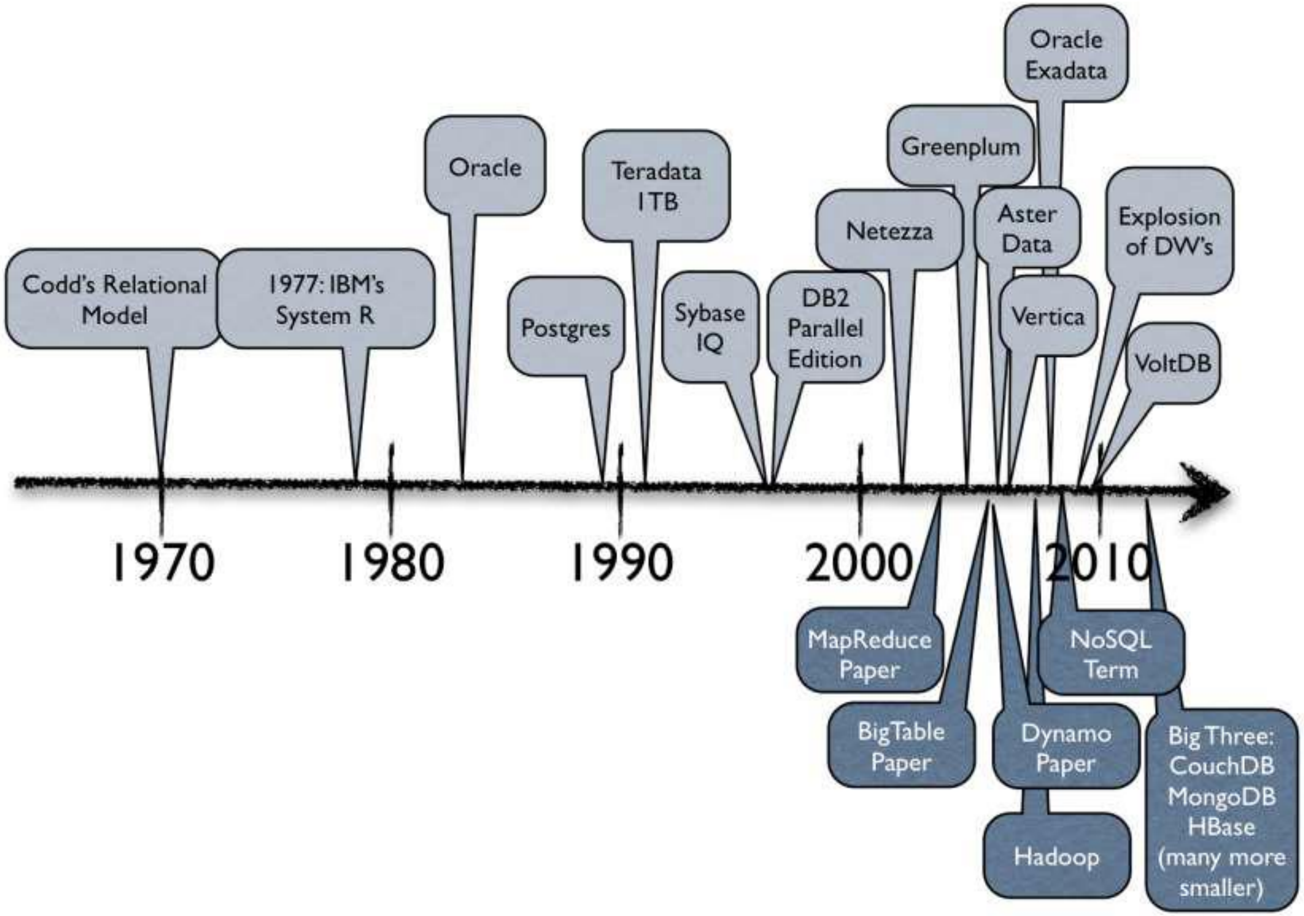
By: Shahab Safaee & Morteza Zahedi
Software Engineering PhD
Email: safaee.shx@gmail.com , morteza.zahedi.a@gmail.com

cibtrc.ir
cibtrc
cibtrc

Agenda

- Some history
- Relational databases
- ACID Theorem
- Scaling Up
- Distributed Database Systems
- CAP Theorem
- What is NoSQL?
- BASE Transactions
- NoSQL Types
- Some Statistics
- NoSQL vs. SQL Summery

A brief history of databases



Relational databases

- Benefits of Relational databases:
  - Designed for all purposes
  - ACID
  - Strong consistency, concurrency, recovery
  - Mathematical background
  - Standard Query language (SQL)
  - Lots of tools to use with i.e: Reporting, services, entity frameworks, ...
  - Vertical scaling (up scaling)

ACID Theorem

- **A**tomic:
  - All of the work in a transaction completes (commit) or none of it completes
  - All operations in a transaction succeed or every operation is rolled back.
- **C**onsistent:
  - A transaction transforms the database from one consistent state to another consistent state. Consistency is defined in terms of constraints.
  - On the completion of a transaction, the database is structurally sound.
- **I**solated:
  - The results of any changes made during a transaction are not visible until the transaction has committed.
  - Transactions do not contend with one another. Contentious access to data is moderated by the database so that transactions appear to run sequentially.
- **D**urable:
  - The results of a committed transaction survive failures
  - The results of applying a transaction are permanent, even in the presence of failures.

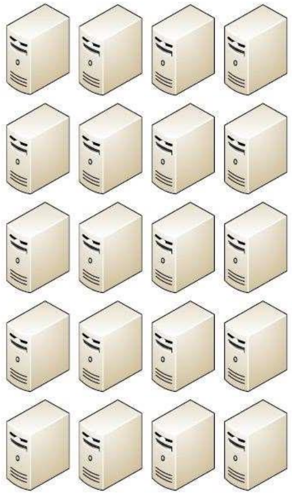
Era of Distributed Computing

But…

- Relational databases were not built for **distributed applications.**

Because…

- Joins are expensive
- Hard to scale horizontally
- Impedance mismatch occurs
- Expensive (product cost, hardware , Maintenance)

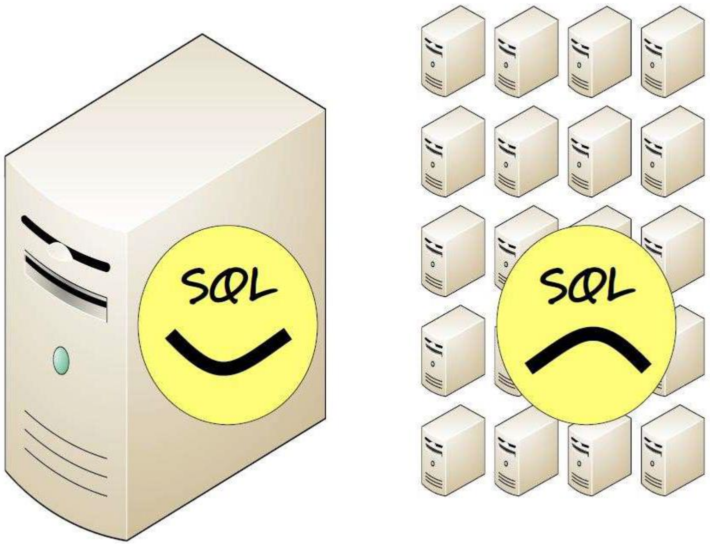


Era of Distributed Computing

But…
- Relational databases were not built for **distributed applications.**

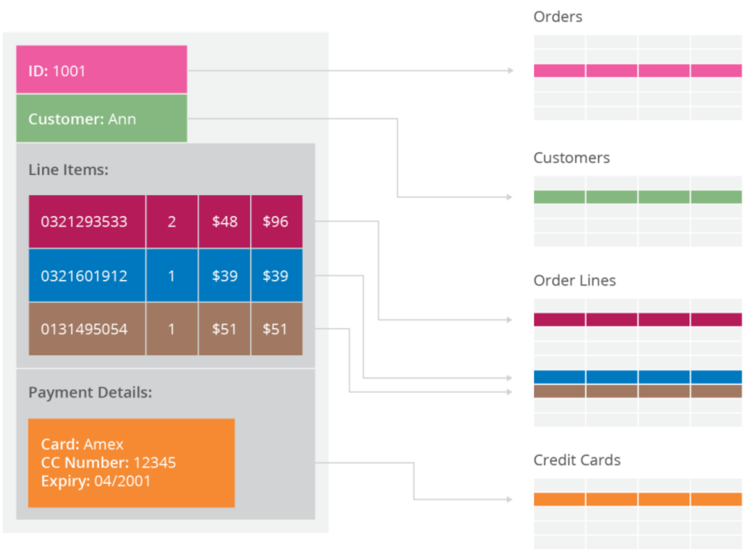
Because…
- Joins are expensive
- Hard to scale horizontally
- Impedance mismatch occurs
- Expensive (product cost, hardware , Maintenance)

And …
It's weak in:
- Speed (performance)
- High availability
- Partition tolerance



Scaling Up

- Issues with scaling up when the dataset is just too big
- RDBMS were not designed to be distributed
- Began to look at multi-node database solutions
- Known as ‘scaling out’ or ‘horizontal scaling’
- Different approaches include:
  - Master-slave
  - Sharding



Scaling RDBMS – Master/Slave

- Master-Slave
  - All writes are written to the master. All reads performed against the replicated slave databases
  - Critical reads may be incorrect as writes may not have been propagated down
  - Large data sets can pose problems as master needs to duplicate data to slaves

Scaling RDBMS - Sharding

- Partition or sharding
  - Scales well for both reads and writes
  - Not transparent, application needs to be partition-aware
  - Can no longer have relationships/joins across partitions
  - Loss of referential integrity across shards

Sharding Advantages

- Tables are divided and distributed into multiple servers
- Reduces index size, which generally improves search performance
- A database shard can be placed on separate hardware
- greatly improving performance
- if the database shard is based on some real-world segmentation of the data then it may be possible to infer the appropriate shard membership easily and automatically

Other ways to scale RDBMS

- Multi-Master replication
- INSERT only, not UPDATES/DELETES
- No JOINs, thereby reducing query time
  - This involves de-normalizing data
- In-memory databases

What we need?

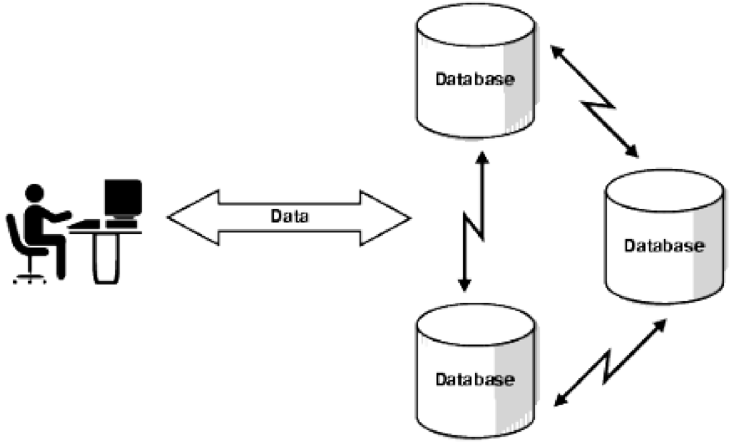
We need a distributed database system having such features:

- **High Concurrency**
- **High Availability**
- **Fault tolerance**
- **High Scalability**
- **Low latency**
- **Efficient Storage**
- **Reduce Manage and Operation Cost**

Which is impossible!!!
According to CAP theorem

Distributed Database Systems

- Data is stored across several sites that share no physical component.
- Systems that run on each site are independent of each other.
- Appears to user as a single system.



Distributed Data Storage

- **Partitioning :**
  - Data is partitioned into several fragments and stored in different sites.
  - Horizontal – by rows.
  - Vertical – by columns.

- **Replication :**
  - System maintains multiple copies of data, stored in different sites.

Replication and Partitioning can be combined !

Partitioning

key	value
x	5
y	7
z	10
w	12

A

key	value
x	5

B

key	value
y	7
z	10

C

key	value
w	12

Horizontal

A

key
x
y
z
w

B

value
5
7
10
12

Vertical

- Locality of reference – data is most likely to be updated and queried locally.

Replication

key	value
x	5
y	7
z	10
w	12

A

key	value
x	5
y	7

B

key	value
x	5
z	10

C

key	value
y	7
w	12

D

key	value
z	10
w	12

- **Pros** – Increased availability of data and faster query evaluation.
- **Cons** – Increased cost of updates and complexity of concurrency control.

CAP Theorem

- In 2000, Berkeley, CA, researcher Eric Brewer published his now foundational CAP Theorem
  - (consistency, availability and partition tolerance)
- which states that it is impossible for a distributed computer system to simultaneously provide all three CAP guarantees.
- In May 2012, Brewer clarified some of his positions on the oft-used "two out of three" concept.

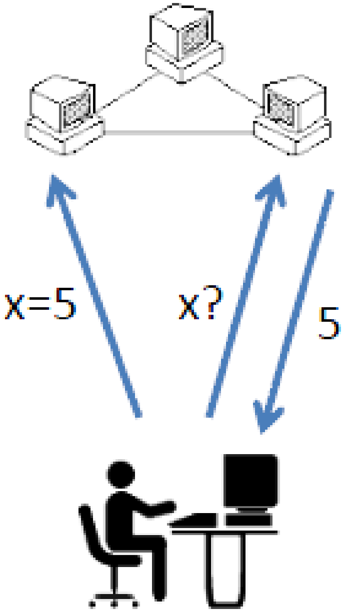
CAP Theorem

- **Consistency:**
  - all nodes see the same data at the same time
- **Availability:**
  - a guarantee that every request receives a response about whether it was successful or failed
- **Partition tolerance:**
  - the system continues to operate despite arbitrary message loss or failure of part of the system

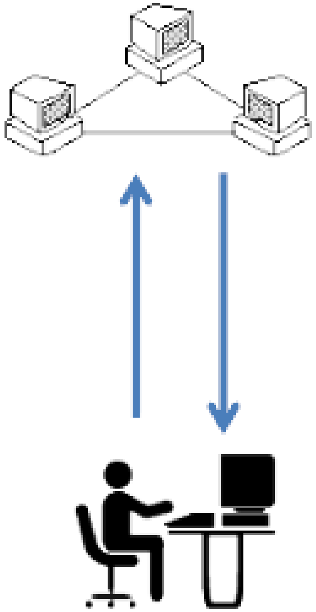
Theorem – You can have at most two of these properties for any shared-data system.

CAP Theorem

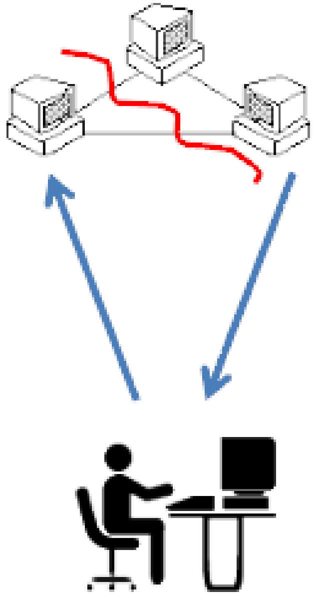
Consistency



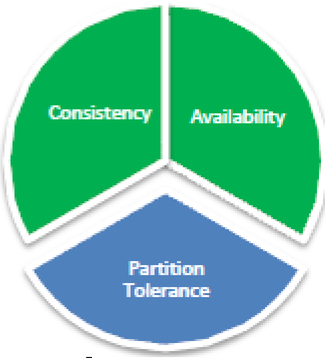
Availability



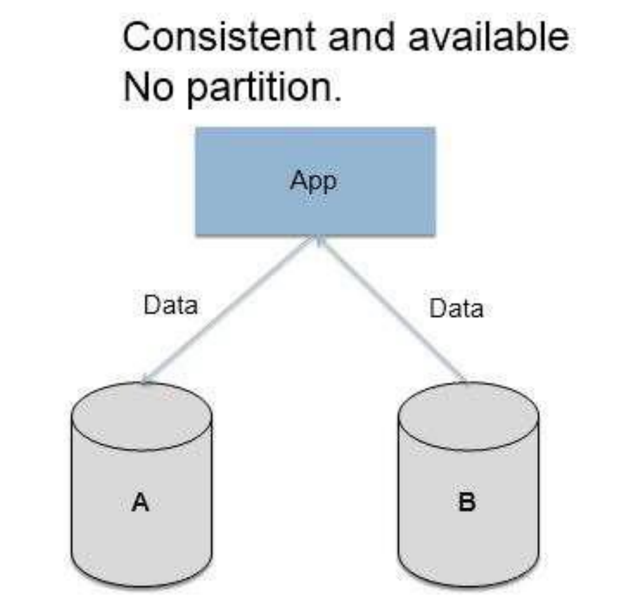
Partition tolerance



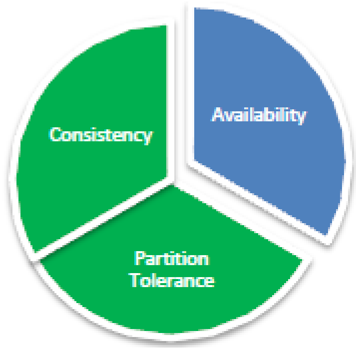
CAP - 2 of 3



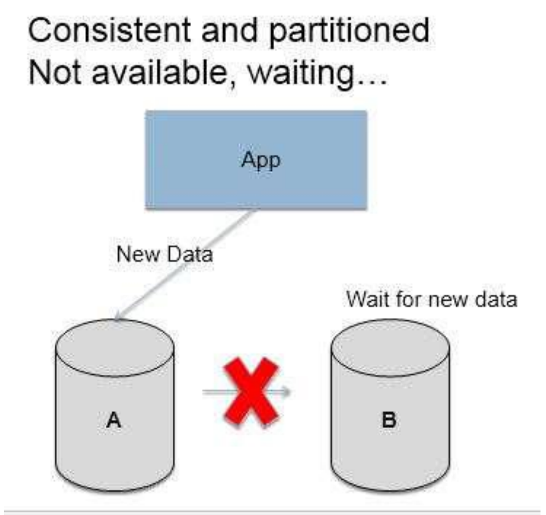
- If there are no partitions, it is clearly possible consistent, available data (e.g. read-any write-all).Best-effort availability:
- Examples:
  - RDBMs



CAP - 2 of 3

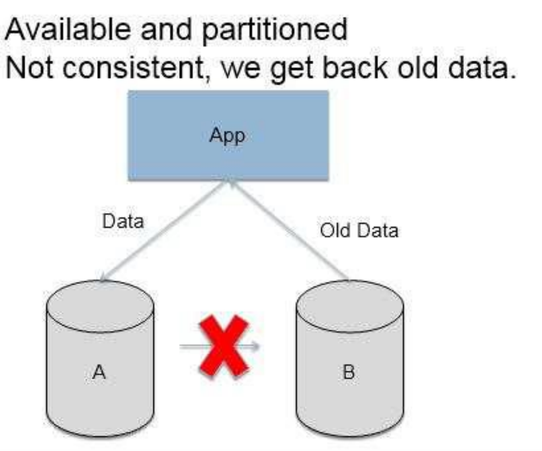


- Trivial:
  - The trivial system that ignores all requests meets these requirements.
- Best-effort availability:
  - Read-any write-all systems will become unavailable only when messages are lost.
- Examples:
  - Distributed database systems, BigTable



CAP - 2 of 3

- Trivial:
  - The service can trivially return the initial value in response to every request.
- Best-effort consistency:
  - Quorum-based system, modified to time-out lost messages, will only return inconsistent(and, in particular, stale) data when messages are lost.
- Examples:
  - Web cashes, Dynamo



Reference

- http://nosql-database.org/
- http://wikibon.org/wiki/v/21_NoSQL_Innovators_to_Look_for_in_2020#Introduction
- https://db-engines.com
- http://basho.com/posts/technical/why-vector-clocks-are-easy/
- …


Thank
You
Merci
Gracias
Arigatô
Hvala
Danke
Spacibo
KamSahHamnida
TerimaKasih
Mèsi
Xièxie
Takk
aDank
BarakAllahuFiik
Grazie
Chokrane
Kiitos
Dziakuju
GratiasAgimus
GratiasAgo
Motashakkeram
TänanVäga
Tak
Dankie
Gracie
CámOn
Multumesc
Saha
Obrigado
Salamat
Bayarlalaa
Nirringrazzjak
NajjisTuke
Efharisto
Faleminderit
Gràcies
Dakujem
Taiku
Mercé
Matondo
Blagodaram
ThintKo
Buznyg
Rahmat
TapadhLeat
Köszönöm
Murakoze
WaadMahadsantahay
Tanemirt
Tenki
Grandmercé
KopKhunKha
Sagolun
Chnorakaloutioun
Aabhar
Dhanyavaad
Barkal
Grazzi
Mauruuru
Ačiū
Gratzias
KhobChaiDeu
Mamnun
aNiKié
Shukriya
Enkosi
Ngiyabonga
TapadhLeibh
AsanteSana
Waita
Welalin
TangioTumas
Dhanyavad
Sukriya
KeYaLeboha
Vinaka
Dankon
Barka
Toda
Mersi
Dhanyavadagalu
Blagodaria
Nandri
Merkzi
Akun
Mahalo
Gracinas
DankJuWel
ÇoxSagOlun
Meharbani
Nizzik ajr
Aguyjé
Sulpáy
Dhanyavadalu
Diolch
EskerrikAsko Dankewol
Spas
Wado
Marahaba
Diky
TakkFyri
Miigwetch
Asante
Rahmet
MaloTeOfa
Gràçie
Akiba
Tanmirt
Akpé
Misaotra
Trugéré
ANiKié
DjiereDieuf
Paldies
TeşekkürEderim
Sobodi
AabhariAahe
Yekeniele
KurreSumanga
KyayTzuTinPaTe
GoRaibhMaithAgat
Mochchakkeram
Dhanyabaad
KopKhunKrap
Tau
Nanni
Miletker
Trugarez
Bedankt
Zikomo
M-Sapo
FaafetaiLava
Nouari
TesekkurEdirem
DankJe