Databases : Lecture 11 : Beyond ACID/Relational databases

Timothy G. Griffin
Lent Term 2014

- Rise of Web and cluster-based computing
- "NoSQL" Movement
- Relationships vs. Aggregates
- Key-value store
- XML or JSON as a data exchange language
- Not all applications require ACID
- CAP = Consistency, Availability, and Partition tolerance
- The CAP theorem (pick any two?)
- Eventual consistency

Apologies to Martin Fowler ("NoSQL Distilled")

Application-specific databases have always been with us . . .

Two that I am familiar with:

Daytona (AT&T): "Daytona is a data management system, not a database". Built on top of the unix file system, this toolkit is for building application-specific and highly scalable data stores. Is used at AT&T for analysis of 100s of terabytes of call records. http://www2.research.att.com/~daytona/

But these systems are **proprietary**.

Open source is a hallmark of NoSQL

DataBlitz (Bell Labs, 1995) : Main-memory database system designed for embedded systems such as telecommunication switches. Optimized for simple key-driven queries.

What's new? Internet scale, cluster computing, open source . . .

Something big is happening in the land of databases

The Internet
+ cluster computing
+ open source systems
➡
many more points in the database design space are being explored and deployed

Broader context helps clarify the strengths and weaknesses of the standard relational/ACID approach.

http://nosql-database.org/

NoSQL

Your Ultimate Guide to the Non - Relational Universe!

[the best

...neve

News

NoSQL DEFINITION:Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable.

The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly. Often more characteristics apply such as: schema-free, easy replication support, simple API, eventually consistent / BASE (not ACID), a huge amount of data and more. So the misleading term *"nosql"* (the community now translates it mostly with "not only sql") should be seen as an alias to something like the definition above. [based on 7 sources, 14 constructive feedback emails (thanks!) and 1 disliking comment . Agree / Disagree? Tell me so! By the way: this is a strong definition and it is out there here since 2009!]

LIST OF NOSQL DATABASES [currently 150]

Eric Brewer's PODC Keynote (July 2000)

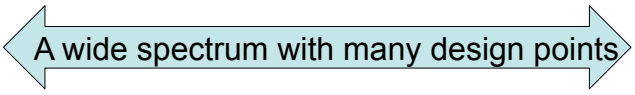
ACID vs. BASE (Basically Available, Soft-state, Eventually consistent)

ACID

- **Strong consistency**
- **Isolation**
- **Focus on "commit"**
- **Nested transactions**
- **Availability?**
- **Conservative (pessimistic)**
- **Difficult evolution (e.g. schema)**

BASE

Weak consistency
Availability first
Best effort
Approximate answers OK
Aggressive (optimistic)
Simpler!
Faster
Easier evolution



"Real internet systems are a careful *mixture* of ACID and BASE subsystems"

The emerging world of databases

This classification is not Complete and is a bit fuzzy-wuzzy. For example, drawing a clear distinction between Key-value stores and Document-oriented databases is not always easy. And this is Rapidly evolving with a lot of cross-fertilization.

Often overlooked in the business-oriented hoopla: This is making BigAnalytics affordable for many scientific efforts (bioinformatics, astronomy, physics, economics,…)

- **Relational**
  - Postgres
  - MySQL
- **Graph databases**
  - Neo4j
  - VertexDB
- **Key-Value stores**
  - Riak
  - Redis
  - BerkeleyDB
- **Column-oriented databases**
  - BigTable,
  - Cassandra
  - Hbase (build on Hadoop)
- **Document-oriented**
  - MongoDB
  - CouchDB

The emerging world of databases

- **Relational**
  - Postgres
  - MySQL
- **Graph databases**
  - Neo4j
  - VertexDB

Attribute-oriented, ACID

- **Key-Value stores**
  - Riak
  - Redis
  - BerkeleyDB
- **Column-oriented databases**
  - BigTable,
  - Cassandra
  - Hbase (build on Hadoop)
- **Document-oriented**
  - MongoDB
  - CouchDB

Aggregate-oriented, Eventual consistency

Aggregates as a natural unit of update

Martin Fowler : "Welcome to the world of polyglot persistence"

More and more we will see data-oriented systems do and will combine traditional Relational DBMS technology with NoSQL technology.

- Must understand what problems each technology solves
- Use right tool for the job

This lecture : I will put emphasis on applications of the form

Traditional RDBMs (normalized/ACID) → Extract → Aggregate-oriented data stores. NoSQL technology

Key-Value Stores

- Mapping Key to blob-of-byte that application must "parse"
  - Example : Riak (modeled on Dynamo, eventual consistency), Cassandra
  - Typically no "query-language" for values
- Mapping Key to "semi-structured" value
  - Example: Redis

Huge advantage: can design data representation so that all data needed for a given update is present on a single machine. Data can easily be partitioned (say by key ranges) over many machines. Map-reduce initiated from set of keys . . .

Disadvantage: Data retrieved by key only. And it is hard to enforce relationships between different values. If this is important for your applications, then perhaps Look elsewhere …

Tables require joins

FK FK
S(A, B, C) ⋈ R(C, D, E) ⋈ T(E, F) (FK = Foreign Key)

A	B	C	D	E	F
A1	B1	C1	D1	E1	F1
A1	B1	C1	D2	E2	F2
A1	B1	C1	D3	E3	F3
A2	B2	C2	D4	E4	F4
A2	B2	C2	D5	E5	F5
. . .	. . .				
. . .					

- How could tables be partitioned over multiple servers?
- Enforcing referential integrity is VERY difficult in a distributed database

The Key-value approach

FK FK
S(A, B, C) ⋈ R(C, D, E) ⋈ T(E, F) (FK = Foreign Key)

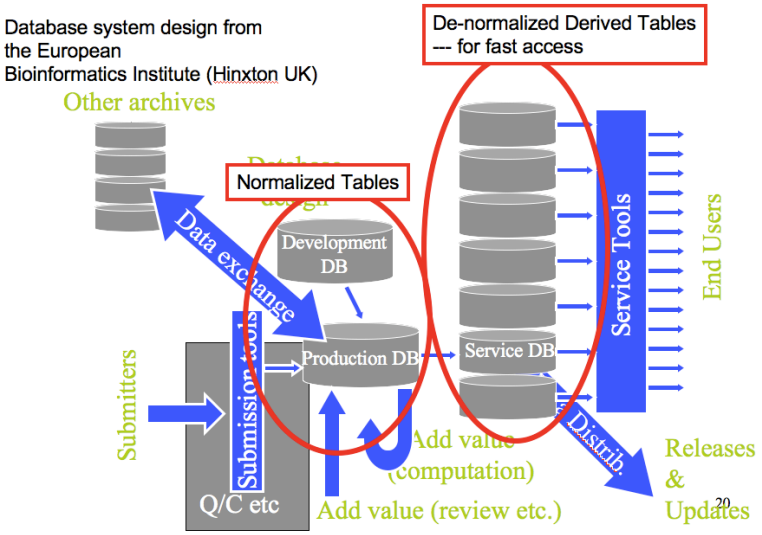
A	B	C	D	E	F
A1	B1	C1	D1	E1	F1
A1	B1	C1	D2	E2	F2
A1	B1	C1	D3	E3	F3
A2	B2	C2	D4	E4	F4
A2	B2	C2	D5	E5	F5
. . .	. . .				
. . .					

Use this instead

```
{A : A1,
 B : B1,
 stuff : [
    {D : D1, F: F1},
    {D : D2, F: F2},
    {D : D3, F: F3}
 ]
}
```

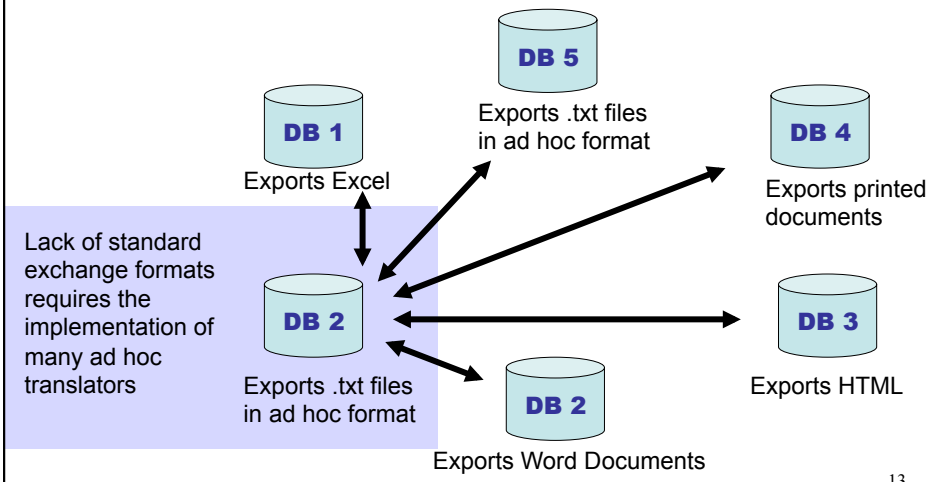
The collection of JSON objects (keyed on A) is horizontally partitioned (sharded) across many servers. When accessed, all of the application's data is in one object.

Example from Lecture 1



Document-oriented systems can be to manage the RDBMS "Publishing Problem"

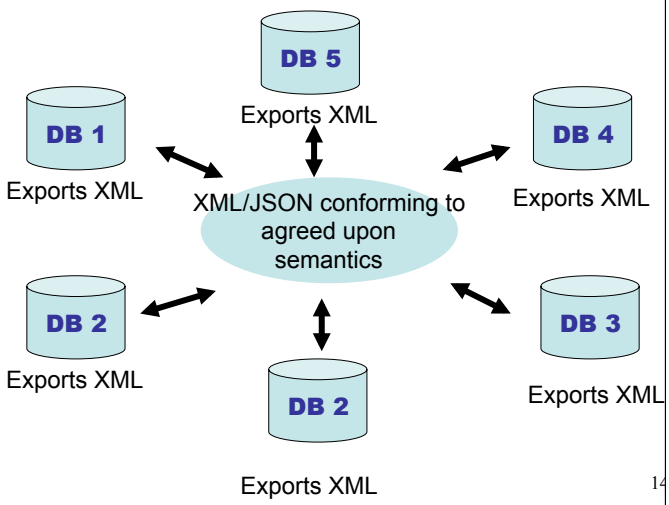
Need to share data without exposing internal details of your database.



13

XML (or JSON) as a data exchange format

Using document-oriented NoSQL software for data exchange is an attractive option.



14

Examples of domain specific XML DTDs (similar developments with JSON)

- There are now lots of DTDs that have been agreed by groups, including
  - WML: Wireless markup language (WAP)
  - OFX: Open financial exchange
  - CML: Chemical markup language
  - AML: Astronomical markup language
  - MathML: Mathematics markup language
  - SMIL: Synchronised Multimedia Integration Language
  - ThML: Theological markup language

15

Fallacies of Distributed Computing (Peter Deutsch)

Essentially everyone, when they first build a distributed application, makes the following eight assumptions. All prove to be false in the long run and all cause big trouble and painful learning experiences.

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

https://blogs.oracle.com/jag/resource/Fallacies.html

Brewer's CAP conjecture (2000)

- **C**onsistency
- **A**vailability
- **P**artition tolerance

Conjecture :
You can have at most two.

A formal proof:

Nancy Lynch and Seth Gilbert,
"Brewer's conjecture and the feasibility
of consistent, available, partition-tolerant web services",
ACM SIGACT News, Volume 33 Issue 2 (2002), pg. 51-59.

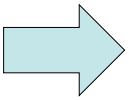
But what do the CAP terms really mean? There seems to be no consensus . . .

Random samples of various definitions found in the literature …

- **C**onsistency
  - The system can guarantee that once you store a state in the system, it will report the same state in every subsequent operation until the state is explicitly changed by something outside the system.
  - Is equivalent to having a single up-to-date copy of the data
- **A**vailability
  - All clients can find some replica of the data, even in the presence of failures
  - A guarantee that every request receives a response about whether it was successful or failed
- **P**artition tolerance
  - The system properties hold even when the system is partitioned
  - The system continues to operate despite arbitrary message loss or failure of part of the system

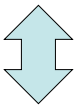
Pick any two? A better formulation.

Suppose you have a highly distributed system



then you must engineer trade-offs between

Consistency



Availability