Trends & needs in current computing

- **Adapt to changing requirements in a fast changing business and IT environment**
- **Blend data from various sources (data integration) on the fly and answer complex queries in databases with heterogeneous structure**
- **Size of databases impacts on processing speed – support new processing approaches**
- **Dynamic (streamed) contents**
- **Distribute data through multiple servers and locations (Distributed databases)**
- **Discover relationships in databases of heterogeneous documents or objects**
- **Approximate results: matches and ranking**

Big Data

Big Data – V V V

- **Volume: large amounts of data.**
  - beyond what can reasonably fit into a single memory/ hard drive
- **Velocity: data loaded/updated at a high rate**
  - E.g., streaming data from sensors, from social network feeds, at the stock exchange, created on the Web…
  - Delays are costly
- **Variety/Variability: heterogeneity of schemas, formats, inconsistentcy,**
  - Need approaches that allow for easy adaptation in what is stored and how it is processed
- **… Some also speak of Veracity (how trusted the data can be, data quality)**

Scalability

- **For Volume and Velocity we need Scalable solutions:**
  - systems that can grow at least proportionally to the needs. (recall comp complexity)
- **Purpose-build computers are $$$ (scale up, vertical scalability)**
- **Ability to add more small/cheap resources as requirements grow (scale out, horizontal scalability);**
- **Embraces failure by redundancy**
- **Consequence: databases are partitionned between multiple physical systems (computers).**

Distributed Databases

Partitioning (~clustering)

- **Logical (Share everything):**
  - Can reside on the same server
  - Shares CPU, Memory, Disk
- **Physical:**
  - Resides on different servers/computing environments
  - Each partition has its own CPU, Memory, Storage
  - Requires network communication (low-latency)
  - One row / data unit is only on a subset of partitions.
- **Cluster: Connected physical servers of the same "kind" with mutually balanced loads**

Data partitioning

- **Horizontal data partition – (partition of rows by key value range)**
  - Also known as ***sharding*** – different rows in different partitions (and likely on different servers)
  - Results may require UNIONS
- **Vertical data partitioning**
  - Normalization
  - Row splitting
  - Requires joins…

Sharding

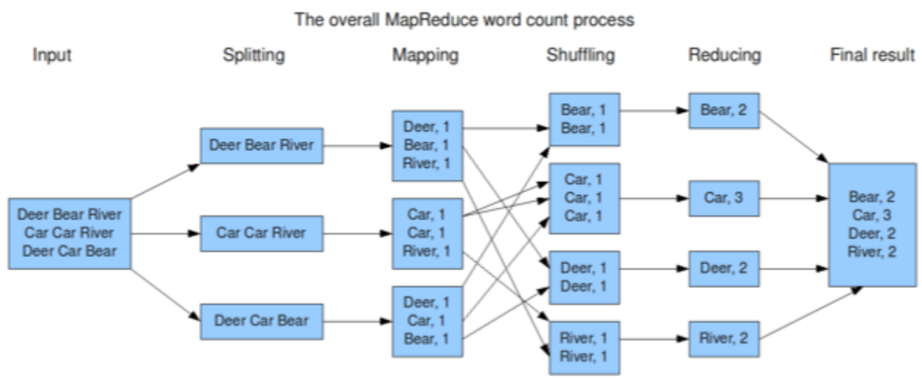
- **Sharding ranges can be grounded in reality**
  - E.g., By preference, store similar/close values together
  - Store values likely accessed from near a given cluster together
  - Geo - sharding – using location of data entries to shard:

    Store all US restaurant data entries in US cluster located in US, and EU restaurant data entries in Europe – assume more users require near restaurants

Processing of data in partitions

- **Partitioned data require/support different computing approaches**
- **Suited for parallel computation instead of sequential processing**
- **MapReduce is an example (but not designed primarily for DB...)**
- **Two step:**
  - Map: filters/sorts data by some attributes - paralleliseable atomic computation if(x > 1) then a else b
  - Reduce: summary operation mean[a], count[a],...
- **Independence between map operations enable running on sharded data storage**

Map-Reduce



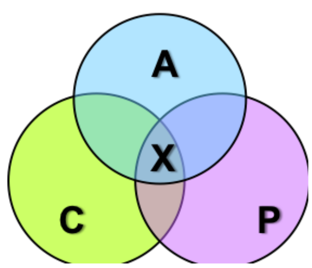
Partitioning consequences

- **Communication overhead to assure database consistency after transactions – during synchronisatio or replication process.**
- **In physically distributed DBs network latency matters** (10km = 67µs → cache write =10-20µs!)
- **Simultaneous updates may occur**
- **Locking – concurrency control mechanism that assures exclusive access to a resource (value, row, table…)**
- **BUT! Locking impacts on availability (the resource closes to users when updated [locked])**

Brewer's CAP Theorem

- **Consistency –** A read is guaranteed to return the most recent value to any client.
- **Accessibility –** guarantee that every request to a non-failing node receives a response about whether it succeeded or failed
- **Partitioning (partition tolerance) –** the system continues to operate despite arbitrary message loss or failure of part of the system when network partition occurs.

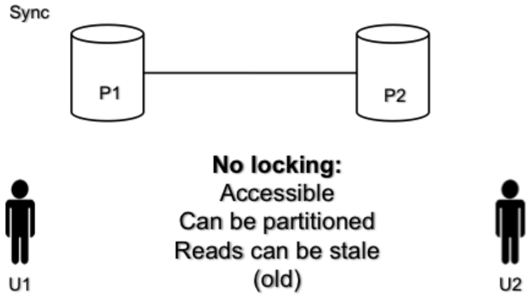
All three can not be met



CAP illustration – Consistency violation

→ AP system

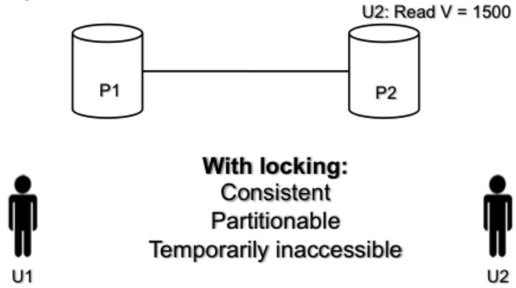
1.	V:	1000	
2.	Sync		
3.	V:	1000	1000
4.	U1:	V+500 = 1500	
5.			U2: Read V = 1000
6.	Sync		



CAP illustration – Availability violation

→ CP system

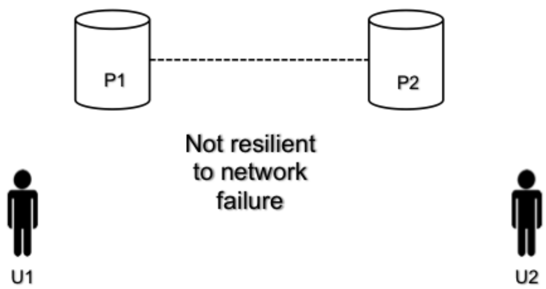
1.	V:	1000	
2.	Sync		
3.	V:	1000	1000
4.	U1:	V+500 = 1500	Lock
5.	Sync		
6.			U2: Read V = 1500



CAP illustration – Partition Tolerance violation

→ CA system

When a network is partitioned, all messages sent from nodes in one component of the partition to nodes in another component are lost. A CA distributed system does not exist (Utopia)



Solutions?

- ACID solution: PAXOS protocol (consensus or quorum based techniques) and others → NewSQL databases
- Basically Available, Soft state, Eventual consistency

Ultimately (eventually) all reads will be the same (will converge). Until then, not → you may get *stale* values.

Your browser showing an old Website, or you Gmail account when offline – you need to refresh the cache (because the cache is a distributed DB). Business solution: tell users to refresh!

NoSQL

- Data storage model OTHER THAN the relational model…
- No single unifying concept, and usually a specialised application area → there is no one size fits all!
- Typically no good at joins
- No shared query language
- Use of APIs and diverse programming languages to interact
- Often Data = resources → REST interaction over HTTP

REST & DBs (an aside)

- HTTP – hypertext transfer protocol powering the Web
- REST: Representational Stateless Transfer
- Resource identified by URL
- 4 actions: POST,GET,PUT,DELETE (~CRUD)
- Standardised response codes (200: OK, 404: Not found)
- Document type (request/response) = representation

NoSQL – typology

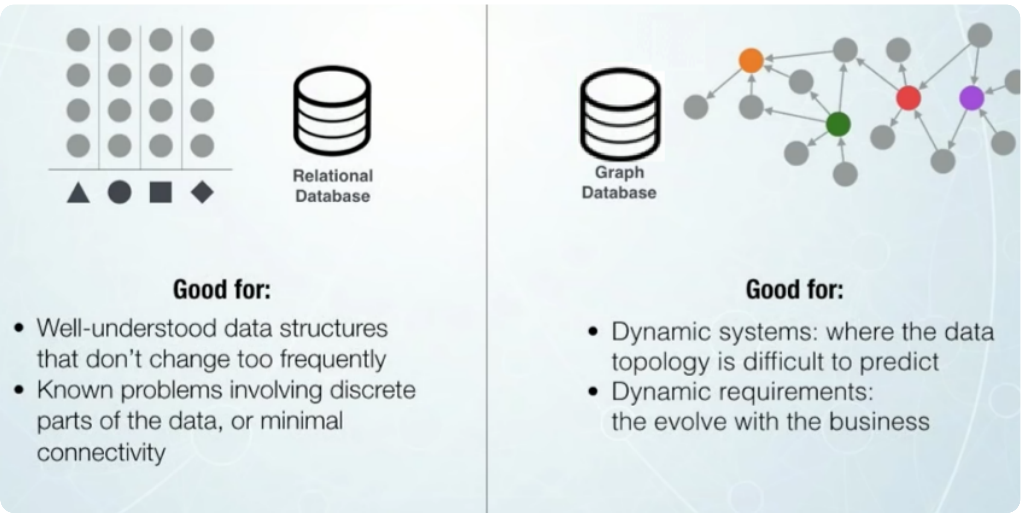
- Key-Value stores: a unique key and an attached value (opaque). Often used for cache. HStore in Postgres
- Document store: values are not opaque: structured format (XML, JSON). Can be indexed by full text or some attributes (JSON). Good for semi-structured content. JSON data type in Postgres
- Column DBs: focus is on attribute-first indexing (as opposed to row-first). Optimised for column aggregates over identical entities (sums, averages,…)

NoSQL – typology II

- Graph DB: Labeled property graphs. Nodes:: entities, Edges :: relationships. Great flexibility to model relationships and their properties and derive new ones. Useful for Linked Data. Do not require joins as these can be explicitly stored. Unified query langage based on Cypher (Neo4J) coming.
- Array DB: special DBs for raster data (arrays of simple values) in a regular grid of *n* dimensions. Important for Earth Sciences, Meteorology, RS, Terrains and other scientific apps.
- OLAP Cubes: hierarchical, n-dimensional value storage. Important in social sciences et al.

Graph databases

- Graph databases: modelling richly-connected data and querying interactions among the data.



a Document DB – CouchDB

Versioning: Couch transactions

Uses revisions – each doc has a revision (rev) number

- **Retrieve the document, take note of the _rev property that CouchDB sends along**
- **Do operation….**
- **Send the updated document back, using the _rev property (PUT)**
- **If the _rev matches the currently stored number - done!**
- **If there's a conflict (when _rev doesn't match), retrieve the newest document version**

Probabilistic databases – handling uncertainty

- **Think of sensor networks – probability that something is what it reports**
- **Case 1: Uncertain values (inputs)**
  - Attribute uncertainty (temperature +-.5 deg C, position uncertainty)
  - Correlated uncertainty: correlation of x and y in geo coordinates
- **Case2: Approximate query answers (outputs)**
  - When you query on Google and nothing matches (typo?)
  - When you search on Domain for a flat and nothing is perfect
  - Rankings are a typical solution

- **A probabilistic database considers a discrete probability distribution of all possible *Worlds*.**
- **A relational database explicitly models only one of such worlds.**
- **The probability of all the worlds sums to 1.**

PDB = Example

Customer	Address	Product
John	Seattle	Gizmo
John	Seattle	Camera
Sue	Denver	Gizmo

$Pr(I_1) = 1/3$

Customer	Address	Product
John	Boston	Gadget
Sue	Denver	Gizmo

$Pr(I_2) = 1/12$

Customer	Address	Product
John	Seattle	Gizmo
John	Seattle	Camera
Sue	Seattle	Camera

$Pr(I_3) = 1/2$

Customer	Address	Product
John	Boston	Gadget
Sue	Seattle	Camera

$Pr(I_4) = 1/12$

Possible worlds = $\{I_1, I_2, I_3, I_4\}$

15

…age: Dan Suciu, UoW Probability DB lectures

Research challenges (of interest to us)

- ***Self-Healing Maps: ML in databases for error detection and rectification, incl. outlier detection**
- ***Place databases – Graph databases (Property graphs) and spatial operations/indices and semantic reasoning**
- **High performance mobile object analytics with MObj databases (incl mapmatching, characterization, viz and analytics)**
- **HD maps: database support for high definition maps (autonomous vehicles), db updating**
- **LIDAR/Point cloud data structures and octrees**

Data Quality Assurance

Data Quality

- Fitness for use: can these data be used for a given purpose? (Verzin, 1999)
- There is no absolute "quality".
- Garbage in – garbage out. Data underpin the quality of analytical results.

ISO19157 Spatial Data Quality

- **Components:**
  - Completeness
  - Logical Consistency
  - Positional Accuracy
  - Thematic Accuracy
  - Temporal Quality
  - Usability

Data Quality - traditional approach

- **Traditional approach – e.g., National Mapping Agencies:**
  - Plan the database content for a product,
  - Define collection rules and quality standards
  - Train experts and provide guidelines, require specialist instruments
  - Collect data
  - Process and Evaluate
  - Generate product

- Slow
- Rigid
- No adaptation to spatial variation
- Expensive

Does not scale up!

ata Quality - traditional approach

Assumptions

- What is in the database is always correct, we evaluate new patches for correctness
- World changes slowly
- The destination products are well determined
- Schemas do not change / are constant for significant duration of time

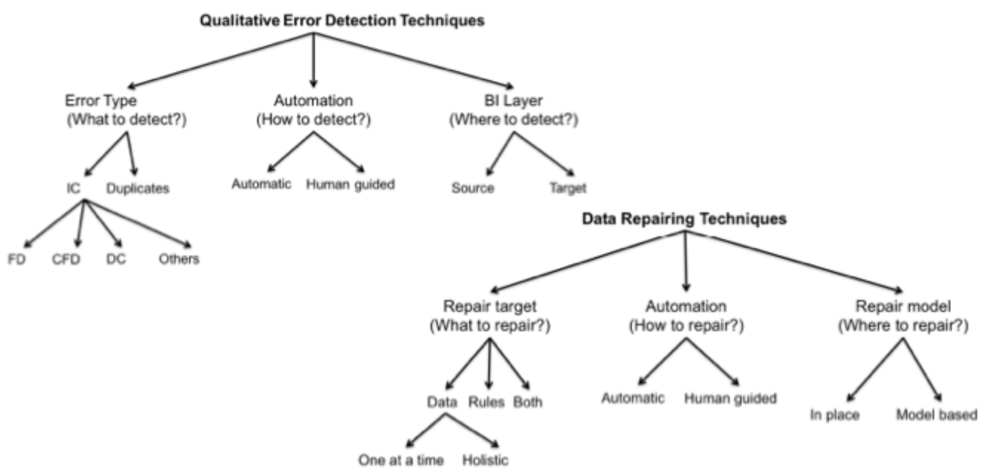
Data Cleaning

- **Largely a manual process**
- **Scripts provided by experts**
- **Underpins data integration**

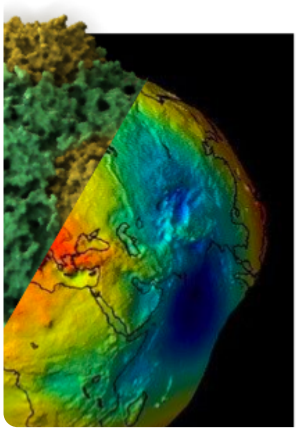
Data Cleaning

- **Error categorization:**
- **Error detection: Ivan**
- **Error rectification: Ra**

Data Cleaning: A Database perspective



Self-Healing Maps



We lack mechanisms enabling responsive and autonomous updates of map data from ad-hoc data sources with guarantees about resulting map integrity and quality

Traditional data collection processes are fit for <u>a</u> purpose, but:

- **Local**: does not adapt well to global needs;
- **Slow**: not suitable for near-real time updates (traffic, changes in the environment);
- **Geared for <u>a</u> well defined** information product;
- **Requires few, well trained experts professional tools,** but loath variation.

Self-Healing Maps

Human body	First line of defence Innate		Specific defence Acquired
	Skin and membranes - If pathogens cannot enter the body, they cannot disrupt it and cause disease	**Non-specific innate responses** - respond the same way upon every infection	Millions of different **antibodies to specific antigenic signals**; after activation **memory cells** confer long-term immunity to a particular pathogen
Spatial database	First line of defence By design, at launch		Specific defence Learned
	DB Schema I want a polygon and you give me a line I want an Int and you give me a String	**Definitions** Captured in extensive, **static** quality assurance scripts, triggers	**Evolving, learned, spatially varying rules** based on observation of manual corrections over time and pattern mining