Database Systems

NoSQL

Examples of database sizes

- Digg: 3 TB – just to store the up/down votes
- Twitter: 7 TB/day
- Facebook:
  - 50 TB – for the private messaging feature
  - 1 PB photos
- eBay: 2 PB data overall

RDBMS weakness

- RDBMSs typically handle ”massive” amounts of data in complex domains, with frequent small read/writes.
  - The archetypical RDBMS serves a bank.
- Cassandra (NoSQL) can perform the ”store” operation into a 50GB database 2500 faster than using MySQL
- Data-intensive applications don’t fit this pattern:
  - MASSIVE+++ amounts of data (e.g. eBay)
  - Super-fast indexing of documents (e.g. Google)
  - Serving pages on high-traffic websites (e.g. Facebook)
  - Streaming media (e.g. Spotify)

Most used DBMS

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

Non-relational databases

- MapReduce framework
  - Google originally; Hadoop (Apache), …
- Key-Value stores
  - BigTable (Google), Cassandra (Apache), …
- Document stores
  - CouchDB, MongoDB, SimpleDB, …
- Graph databases
  - Neo4j, FlockDB, …
- Semi-structured databases
  - (Native) XML databases, …

Semi-structured data (SSD)

- More flexible than the relational model.
  - The type of each ”entity” is its own business.
  - Labels indicate meanings of substructures.
- Semi-structured: it is structured, but not everything is structured the same way!

- Support for XML and XQuery in e.g. Oracle, DB2, SQL Server.

- Special case: Document databases

Document stores

- Roughly: Key-Value stores where the values are ”documents”
  - XML, JSON, mixed semistructured data sets
- Typically incorporate a query language for the document type.
  - See previous lecture for discussion on XML querying.

Document store implementations

- MongoDB
  - Name short for ”Hu**mongo**us”
  - Open source – owned by 10gen
  - JSON(-like) semi-structured storage
  - JavaScript query language
  - Supports MapReduce for aggregations

- Apache CouchDB

SQL vs NoSQL

Terminology and Concepts Many concepts in MySQL have close analogs in MongoDB. This table outlines some of the common concepts in each system.

MySQL	MongoDB
Table	Collection
Row	Document
Column	Field
Joins	Embedded documents, linking

Key-Value Stores

- Key-Value stores is a fancy name for persistant maps (associative arrays, hash tables)
- Extremely simple interface – extremely complex implementations.
- Values can be another {Key-value} documents

NoSQL – Data Example I

3 Entities….

Joins?

Customer

"id": 1,
"timestamp": "2016.03.26-11.47.02.065",
”nid": "B1234455X",
"name": ”Alice",

Factures

”facture": [{
”id": 1,
"date": "26/03/2016",
"total": 6.98
]}

Objects

”objects": [{
”id": 1,
”concept": ”Pencils",
”amount": 3.78}
},
{
”id": 2,
"concept": ”Folder",
”amount": 3.20}
}]

NoSQL – Data Example II

"id": 1,
"timestamp": "2016.03.26-11.47.02.065",
”nid": "B1234455X",
"name": ”Alice",
”facture": [{
”id": 1,
"date": "26/03/2016",
"total": 6.98
”objects": [{
”id": 1,
”concept": ”Pencils",
”amount": 3.78}
},
{
”id": 2,
"concept": ”Folder",
”amount": 3.20}
}]
}]

SQL vs NoSQL

MySQL

```
INSERT INTO users (user_id,
age, status)
VALUES ('bcd001', 45, 'A')
```

```
SELECT * FROM users
```

```
UPDATE users SET status = 'C'
WHERE age > 25
```

MongoDB

```
db.users.insert({
  user_id: 'bcd001',
  age: 45,
  status: 'A'
})
```

```
db.users.find()
```

```
db.users.update(
  { age: { $gt: 25 }
},
  { $set: { status:
'C' } },
  { multi: true }
)
```

SQL vs NoSQL

- Performance
  - NoSQL
    - Denormalized data
    - No JOINs
    - Complex information on a single query
  - SQL
    - Normalized schemas
    - Redundance
    - Complex queries to get complex data

SQL vs NoSQL

- Scaling
  - NoSQL
    - Easy to distribute
    - Easy to spread the data
  - SQL
    - Still a challenge nowadays

Key-Value store implementations

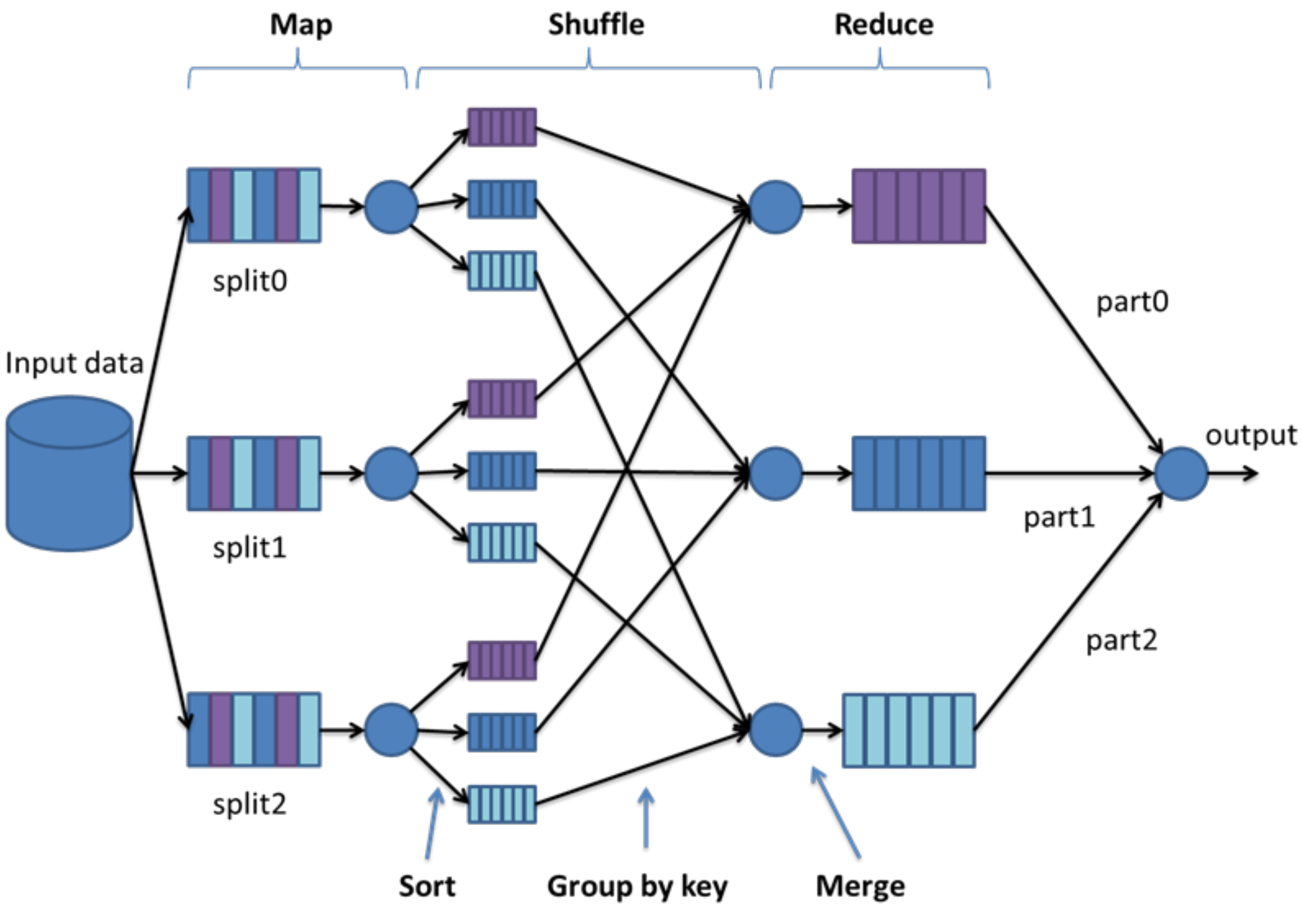
- BigTable (Google)
  - Sparse, distributed, multi-dimensional sorted map
  - Proprietary – used in Google's internals: Google Reader, Google Maps, YouTube, Blogger, …

- Cassandra (Apache)
  - Originally Facebook's PM database – now Open Source (Apache top-level project)
  - Used by Netflix, Digg, Reddit, Spotify, …

MapReduce

- No data model – all data stored in files
- Operations supplied by user:
  - Reader :: file → [input record]
  - Map :: input record → <key, value>
  - Reduce :: <key, [value]> → [output record]
  - Writer :: [output record] → file
- Everything else done behind the scenes:
  - Consistency, atomicity, distribution and parallelism, ”glue”
- Optimized for broad data analytics
  - Running simple queries over all data at once

MapReduce



Map
Shuffle
Reduce
split0
Input data
split1
split2
part0
output
part1
part2
Sort
Group by key
Merge

MapReduce implementations

- The ”secret” behind Google’s success
  - Still going strong.
- Hadoop (Apache)
  - Open Source implementation of the MapReduce framework
  - Used by Ebay, Amazon, Last.fm, LinkedIn, Twitter, Yahoo, Facebook internal logs (~15PB), ...
- MongoDB
- CouchDB

Graph Databases

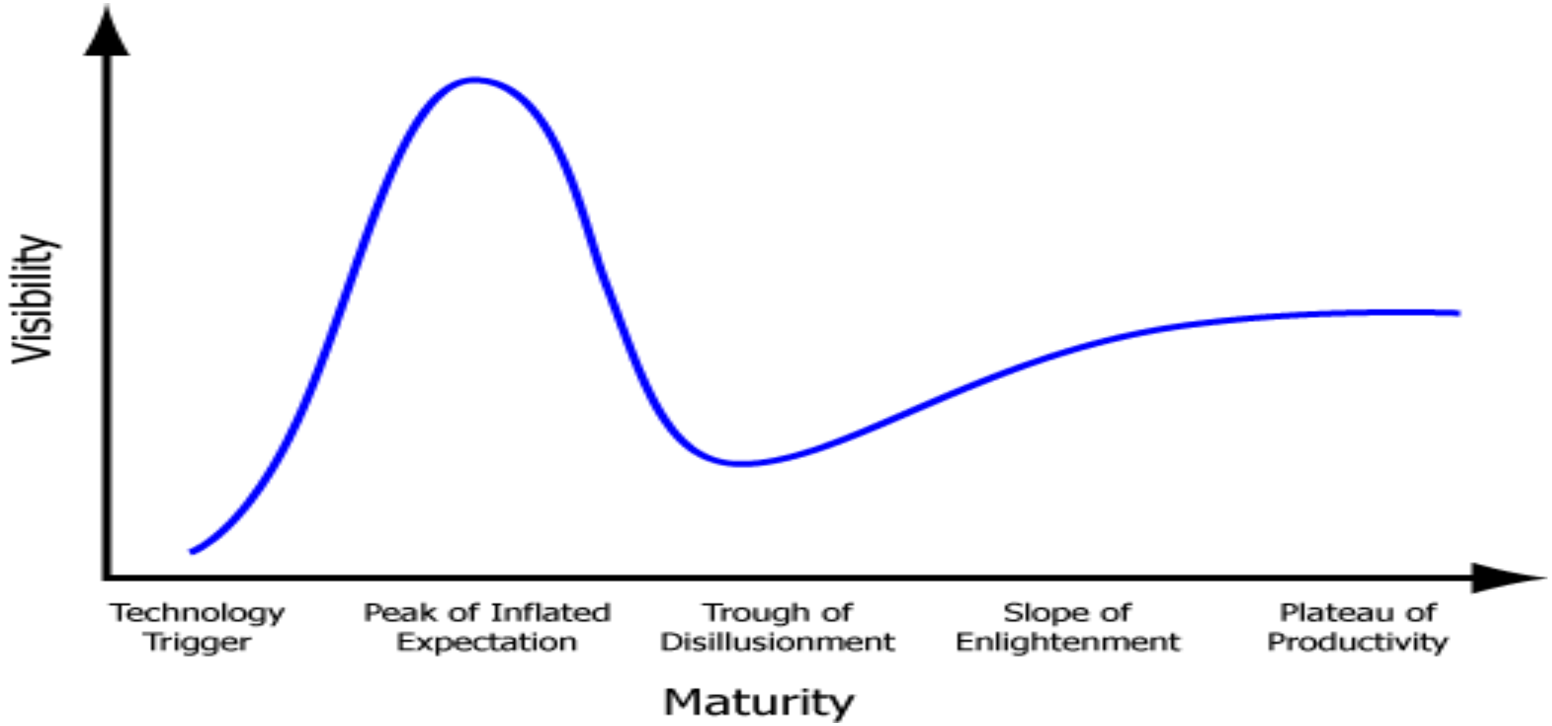
- Data modeled in a graph structure
  - Nodes = ”entities”
  - Properties = ”tags”, attribute values
  - Edges connect
    - Nodes to nodes (relationships)
    - Nodes to properties (attributes)

- Fast access to associative data sets
  - All entities that share a common property
  - Computing association paths

Graph database implementations

- Neo4j
  - Developed in Malmö
  - Specialized query language: Cypher
- FlockDB
  - Initially developed by Twitter to store user relationships
  - Apache licence

NoSQL – a hype?



- NoSQL is not ”the right choice” just because it’s new!
- Relational DBMSs still rule at what they were first designed for: efficient access to large amounts of data in complex domains. That’s still the vast majority!

NoSQL summary

- **Where is SQL ideal?**
  - Requirements can be identified in advance
  - Data integrity is a must
  - Standards-based proven technology.
- **Where is NoSQL ideal?**
  - Unrelated / Indeterminate / evolving data requirements
  - Simpler objectives where time is a requirement
  - Speed and scalability is a must

NoSQL summary

- NoSQL = ”Not only SQL”
- Different data models optimized for different tasks
  – MapReduce, Key-Value stores, Document stores, Graph databases, …
- Typically:

  + efficiency, scalability, flexibility, fault tolerance

  - (no) query language, (less) consistency

NoSQL summary

	NoSQL	SQL
Model	Non-relational	Relational
	Stores data in JSON documents, key/value pairs, wide column stores, or graphs	Stores data in a table
Data	Offers flexibility as not every record needs to store the same properties	Great for solutions where every record has the same properties
	New properties can be added on the fly	Adding a new property may require altering schemas or backfilling data
	Relationships are often captured by denormalizing data and presenting all data for an object in a single record	Relationships are often captured in normalized model using joins to resolve references across tables
	Good for semi-structured, complex, or nested data	Good for structured data
Schema	Dynamic or flexible schemas	Strict schema
	Database is schema-agnostic and the schema is dictated by the application. This allows for agility and highly iterative development	Schema must be maintained and kept in sync between application and database
Transactions	ACID transaction support varies per solution	Supports ACID transactions
Consistency & Availability	Eventual to strong consistency supported, depending on solution	Strong consistency enforced
	Consistency, availability, and performance can be traded to meet the needs of the application (CAP theorem)	Consistency is prioritized over availability and performance
Performance	Performance can be maximized by reducing consistency, if needed	Insert and update performance is dependent upon how fast a write is committed, as strong consistency is enforced. Performance can be maximized by using scaling up available resources and using in-memory structures.
	All information about an entity is typically in a single record, so an update can happen in one operation	Information about an entity may be spread across many tables or rows, requiring many joins to complete an update or a query
Scale	Scaling is typically achieved horizontally with data partitioned to span servers	Scaling is typically achieved vertically with more server resources

The End