CS315: Principles of Database Systems
NoSQL

Arnab Bhattacharya
arnabb@cse.iitk.ac.in

Computer Science and Engineering,
Indian Institute of Technology, Kanpur
http://web.cse.iitk.ac.in/˜cs315/

2nd semester, 2013-14

Tue, Fri 1530-1700 at CS101

NoSQL

- NoSQL is

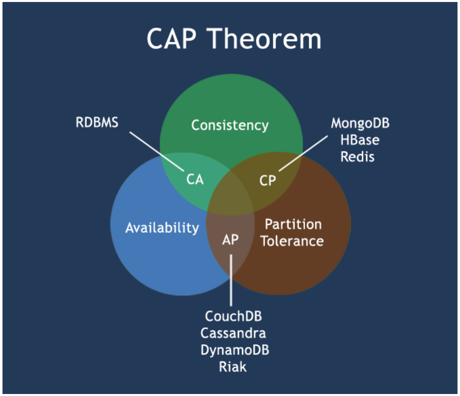
NoSQL

- NoSQL is *not* “no-SQL”
- It is not only SQL
- It does not aim to provide the ACID properties
- Originated as no-SQL though
- Later changed since RDBMS is too powerful to always ignore

NoSQL

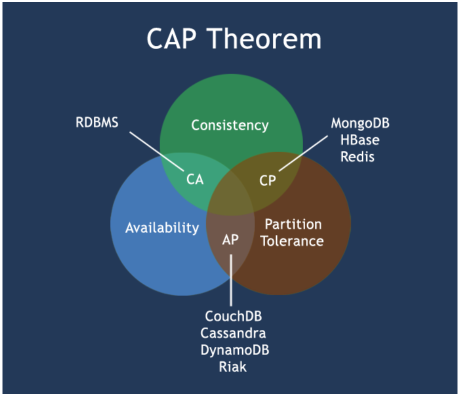
- NoSQL is *not* “no-SQL”
- It is not only SQL
- It does not aim to provide the ACID properties
- Originated as no-SQL though
- Later changed since RDBMS is too powerful to always ignore
- Aims to provide
  - Scalability
  - Flexibility
  - Naturalness
  - Distribution
  - Performance

CAP theorem



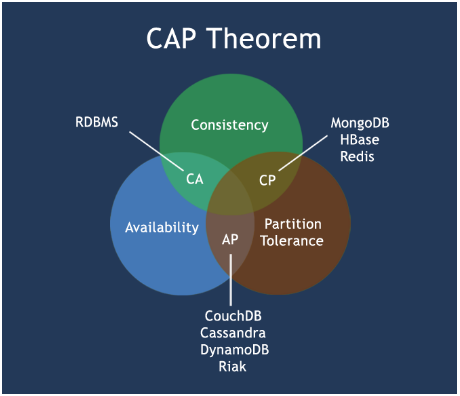
- All of C, A, P cannot be satisfied simultaneously

CAP theorem



- All of C, A, P cannot be satisfied simultaneously
- CA: single-site; partitioning is not allowed
- CP: what is available is consistent
- AP: everything is available but may not be consistent

CAP theorem



- All of C, A, P cannot be satisfied simultaneously
- CA: single-site; partitioning is not allowed
- CP: what is available is consistent
- AP: everything is available but may not be consistent
- Not a theorem – just a hypothesis

BASE properties

- Basically Available: System guarantees availability
- Soft state: State of system is soft, i.e., it may change without input to maintain consistency
- Eventual consistency: Data will be eventually consistent without any interim perturbation

BASE properties

- Basically Available: System guarantees availability
- Soft state: State of system is soft, i.e., it may change without input to maintain consistency
- Eventual consistency: Data will be eventually consistent without any interim perturbation
- Sacrifices consistency

BASE properties

- Basically Available: System guarantees availability
- Soft state: State of system is soft, i.e., it may change without input to maintain consistency
- Eventual consistency: Data will be eventually consistent without any interim perturbation
- Sacrifices consistency
- To counter ACID

Types

- Four main types of NoSQL data stores:
    1. Columnar families
    2. Bigtable systems
    3. Document databases
    4. Graph databases

Columnar storage

- Instead of rows being stored togther, columns are stored consecutively
- A single disk block (or a set of consecutive blocks) stores a single column family
- A column family may consist of one or multiple columns
- This set of columns is called a super column

Columnar storage

- Instead of rows being stored togther, columns are stored consecutively
- A single disk block (or a set of consecutive blocks) stores a single column family
- A column family may consist of one or multiple columns
- This set of columns is called a super column
- Two main types
  - Columnar relational models
  - Key-value stores and/or big tables

Columnar relational models

- *Not* NoSQL and is actually *RDBMS*
- Column-wise storage on the disk

Columnar relational models

- *Not* NoSQL and is actually *RDBMS*
- Column-wise storage on the disk
- Allows faster querying when only few columns are touched on the entire data
- Allows compression of columns
- Provides better memory caching
- Joins are faster since they are mostly on similar columns from two tables

Columnar relational models

- *Not* NoSQL and is actually *RDBMS*
- Column-wise storage on the disk
- Allows faster querying when only few columns are touched on the entire data
- Allows compression of columns
- Provides better memory caching
- Joins are faster since they are mostly on similar columns from two tables
- Is not good for updates
- Is not good when many columns of a few tuples are accessed

Columnar relational models

- *Not* NoSQL and is actually *RDBMS*
- Column-wise storage on the disk
- Allows faster querying when only few columns are touched on the entire data
- Allows compression of columns
- Provides better memory caching
- Joins are faster since they are mostly on similar columns from two tables
- Is not good for updates
- Is not good when many columns of a few tuples are accessed
- Good for OLAP (online analytical processing)
- Not good for OLTP (online transaction processing)

Columnar relational models

- *Not* NoSQL and is actually *RDBMS*
- Column-wise storage on the disk
- Allows faster querying when only few columns are touched on the entire data
- Allows compression of columns
- Provides better memory caching
- Joins are faster since they are mostly on similar columns from two tables
- Is not good for updates
- Is not good when many columns of a few tuples are accessed
- Good for OLAP (online analytical processing)
- Not good for OLTP (online transaction processing)
- Example: MonetDB

Key-value stores

- Two columns: a key and a value
- Key is mostly text
- Value can be anything and is simply an object

Key-value stores

- Two columns: a key and a value
- Key is mostly text
- Value can be anything and is simply an object
- Essentially, actual data becomes “value” and an unique id is generated which becomes “key”

Key-value stores

- Two columns: a key and a value
- Key is mostly text
- Value can be anything and is simply an object
- Essentially, actual data becomes “value” and an unique id is generated which becomes “key”
- Whole database is then just one big table with these two columns
- Becomes schema-less

Key-value stores

- Two columns: a key and a value
- Key is mostly text
- Value can be anything and is simply an object
- Essentially, actual data becomes “value” and an unique id is generated which becomes “key”
- Whole database is then just one big table with these two columns
- Becomes schema-less
- Can be distributed and is, thus, highly scalable
- So, in essence a big distributed hash table

Key-value stores

- Two columns: a key and a value
- Key is mostly text
- Value can be anything and is simply an object
- Essentially, actual data becomes “value” and an unique id is generated which becomes “key”
- Whole database is then just one big table with these two columns
- Becomes schema-less
- Can be distributed and is, thus, highly scalable
- So, in essence a big distributed hash table
- All queries are on keys
- Keys are necessarily indexed
- Uses memcache where most queried keys are persisted in memory for faster access

Key-value stores

- Two columns: a key and a value
- Key is mostly text
- Value can be anything and is simply an object
- Essentially, actual data becomes "value" and an unique id is generated which becomes "key"
- Whole database is then just one big table with these two columns
- Becomes schema-less
- Can be distributed and is, thus, highly scalable
- So, in essence a big distributed hash table
- All queries are on keys
- Keys are necessarily indexed
- Uses memcache where most queried keys are persisted in memory for faster access
- Example: Cassandra, CouchDB, Tokyo Cabinet, Redis

Bigtable systems

- Started from Google's BigTable implementation
- Uses a key-value store
- Data can be replicated for better availability

Bigtable systems

- Started from Google's BigTable implementation
- Uses a key-value store
- Data can be replicated for better availability
- Uses a timestamp
- Timestamp is used to
    - Expire data
    - Delete stale data
    - Resolve read-write conflicts

Bigtable systems

- Started from Google's BigTable implementation
- Uses a key-value store
- Data can be replicated for better availability
- Uses a timestamp
- Timestamp is used to
  - Expire data
  - Delete stale data
  - Resolve read-write conflicts
- Same value can be indexed using multiple keys
- Map-reduce framework to compute

Bigtable systems

- Started from Google's BigTable implementation
- Uses a key-value store
- Data can be replicated for better availability
- Uses a timestamp
- Timestamp is used to
  - Expire data
  - Delete stale data
  - Resolve read-write conflicts
- Same value can be indexed using multiple keys
- Map-reduce framework to compute
- Example: BigTable, HBase, Cassandra, HyperTable, SimpleDB

Document databases

- Uses documents as the main storage format of data
- Popular document formats are XML, JSON, BSON, YAML
- Document itself is the key while the content is the value
- Document can be indexed by id or simply its location (e.g., URI)

Document databases

- Uses documents as the main storage format of data
- Popular document formats are XML, JSON, BSON, YAML
- Document itself is the key while the content is the value
- Document can be indexed by id or simply its location (e.g., URI)
- Content needs to be parsed to make sense
- Content can be organised further

Document databases

- Uses documents as the main storage format of data
- Popular document formats are XML, JSON, BSON, YAML
- Document itself is the key while the content is the value
- Document can be indexed by id or simply its location (e.g., URI)
- Content needs to be parsed to make sense
- Content can be organised further
- Extremely useful for insert-once read-many scenarios
- Can use map-reduce framework to compute

Document databases

- Uses documents as the main storage format of data
- Popular document formats are XML, JSON, BSON, YAML
- Document itself is the key while the content is the value
- Document can be indexed by id or simply its location (e.g., URI)
- Content needs to be parsed to make sense
- Content can be organised further
- Extremely useful for insert-once read-many scenarios
- Can use map-reduce framework to compute
- Example: MongoDB, CouchDB

Graph databases

- Nodes represent entities
- Edges encode relationships between nodes
- Can be directed
- Can have hyper-edges as well

Graph databases

- Nodes represent entities
- Edges encode relationships between nodes
- Can be directed
- Can have hyper-edges as well
- Easier to find distances and neighbors

Graph databases

- Nodes represent entities
- Edges encode relationships between nodes
- Can be directed
- Can have hyper-edges as well
- Easier to find distances and neighbors
- Example: Neo4J, HyperGraph, Infinite Graph, FlockDB

Discussion

- NoSQL, although started as anti-SQL, is no more so
- More a realisation that for some cases
  - RDBMS does not scale or distribute, and in some other cases
  - ACIDity is an overkill

Discussion

- NoSQL, although started as anti-SQL, is no more so
- More a realisation that for some cases
  - RDBMS does not scale or distribute, and in some other cases
  - ACIDity is an overkill
- NoSQL is not good for every scenario
- Not every good feature reported has been validated
- Most legacy systems still use RDBMS

Discussion

- NoSQL, although started as anti-SQL, is no more so
- More a realisation that for some cases
  - RDBMS does not scale or distribute, and in some other cases
  - ACIDity is an overkill
- NoSQL is not good for every scenario
- Not every good feature reported has been validated
- Most legacy systems still use RDBMS
- NoSQL horizon is shifting rapidly with almost no control or sense
- However, trend is for NoSQL as cloud computing and big data relies on it