NoSQL Data Stores

Ashish Kedia
Final Year, IT

January 29, 2016

Outline

Introduction

- Stands for No-SQL or **N**ot **O**nly SQL

Introduction

- Stands for No-SQL or **N**ot **O**nly SQL
- Class of non-relational data storage systems

Introduction

- Stands for No-SQL or **N**ot **O**nly SQL
- Class of non-relational data storage systems
- Examples include MongoDB, Cassandra, Google Datastore, DynamoDB

Introduction

- Stands for No-SQL or **N**ot **O**nly SQL
- Class of non-relational data storage systems
- Examples include MongoDB, Cassandra, Google Datastore, DynamoDB
- Do not require a fixed table schema

Introduction

- Stands for No-SQL or **N**ot **O**nly SQL
- Class of non-relational data storage systems
- Examples include MongoDB, Cassandra, Google Datastore, DynamoDB
- Do not require a fixed table schema
- Distributed data storage systems

Introduction

- Stands for No-SQL or **N**ot **O**nly SQL
- Class of non-relational data storage systems
- Examples include MongoDB, Cassandra, Google Datastore, DynamoDB
- Do not require a fixed table schema
- Distributed data storage systems
- BASE instead of ACID

Historical Needs

- Large Dataset. Explosion of storage needs

Historical Needs

- Large Dataset. Explosion of storage needs
- Dynamically-typed data with frequent schema changes

Historical Needs

- Large Dataset. Explosion of storage needs
- Dynamically-typed data with frequent schema changes
- High frequency of read-write operations

Historical Needs

- Large Dataset. Explosion of storage needs
- Dynamically-typed data with frequent schema changes
- High frequency of read-write operations
- Horizontal Scalability

Historical Needs

- Large Dataset. Explosion of storage needs
- Dynamically-typed data with frequent schema changes
- High frequency of read-write operations
- Horizontal Scalability
- Rise of Cloud Computing

Historical Needs

- Large Dataset. Explosion of storage needs
- Dynamically-typed data with frequent schema changes
- High frequency of read-write operations
- Horizontal Scalability
- Rise of Cloud Computing
- Spikes in Web Applications

Improvements in RDBMS

- Caching Mechanism - But limited scalability

Improvements in RDBMS

- Caching Mechanism - But limited scalability
- Parallel Databases - OLTP unavailable

Improvements in RDBMS

- Caching Mechanism - But limited scalability
- Parallel Databases - OLTP unavailable
- Write to master, read from slave. Replaced by Paxos

Improvements in RDBMS

- Caching Mechanism - But limited scalability
- Parallel Databases - OLTP unavailable
- Write to master, read from slave. Replaced by Paxos
- Sharding - Loss of references

Improvements in RDBMS

- Caching Mechanism - But limited scalability
- Parallel Databases - OLTP unavailable
- Write to master, read from slave. Replaced by Paxos
- Sharding - Loss of references
- Application needs to be Partition Aware

Improvements in RDBMS

- Caching Mechanism - But limited scalability
- Parallel Databases - OLTP unavailable
- Write to master, read from slave. Replaced by Paxos
- Sharding - Loss of references
- Application needs to be Partition Aware
- Multi-Master, In-Memory, Insert Only, No Join

Brewer's CAP Theorem

- consistency, availability and partitions

Brewer's CAP Theorem

- consistency, availability and partitions
- Availability - 99.999%. Resilient. Large Node System

Brewer's CAP Theorem

- consistency, availability and partitions
- Availability - 99.999%. Resilient. Large Node System
- choose 2 out of 3 for any shared-data system

Brewer's CAP Theorem

- consistency, availability and partitions
- Availability - 99.999%. Resilient. Large Node System
- choose 2 out of 3 for any shared-data system
- to scale, we need partition. So choose A or C

Brewer's CAP Theorem

- consistency, availability and partitions
- Availability - 99.999%. Resilient. Large Node System
- choose 2 out of 3 for any shared-data system
- to scale, we need partition. So choose A or C
- AP Type - Google Datastore, DynamoDB

Brewer's CAP Theorem

- consistency, availability and partitions
- Availability - 99.999%. Resilient. Large Node System
- choose 2 out of 3 for any shared-data system
- to scale, we need partition. So choose A or C
- AP Type - Google Datastore, DynamoDB
- CP Type - Couchbase, Cassandra

Consistency in NoSQL

- determines rules for visibility and apparent order of updates

Consistency in NoSQL

- determines rules for visibility and apparent order of updates
- eventually consistent model

Consistency in NoSQL

- determines rules for visibility and apparent order of updates
- eventually consistent model
- eventually all updates will propagate through the system

Consistency in NoSQL

- determines rules for visibility and apparent order of updates
- eventually consistent model
- eventually all updates will propagate through the system
- BASE as opposed to ACID

Consistency in NoSQL

- determines rules for visibility and apparent order of updates
- eventually consistent model
- eventually all updates will propagate through the system
- BASE as opposed to ACID
- Key-Look ups are always consistent

ACID

- **Atomic** All of the work in a transaction commited or none of it

ACID

- **Atomic** All of the work in a transaction commited or none of it
- **Consistent** Consistency is defined in terms of constraints. Always obeyed.

ACID

- **Atomic** All of the work in a transaction commited or none of it
- **Consistent** Consistency is defined in terms of constraints. Always obeyed.
- **Isolated** Results are not visible until the transaction has committed.

ACID

- **Atomic** All of the work in a transaction commited or none of it
- **Consistent** Consistency is defined in terms of constraints. Always obeyed.
- **Isolated** Results are not visible until the transaction has committed.
- **Durable** The results of a committed transaction survive failures

BASE

- Acronym contrived to be the opposite of ACID

BASE

- Acronym contrived to be the opposite of ACID
- **B**asically **A**vailable,

BASE

- Acronym contrived to be the opposite of ACID
- **B**asically **A**vailable,
- Soft state,

BASE

- Acronym contrived to be the opposite of ACID
- **B**asically **A**vailable,
- Soft state,
- Eventually Consistent

BASE

- Acronym contrived to be the opposite of ACID
- **B**asically **A**vailable,
- Soft state,
- Eventually Consistent
- Weak consistency stale data OK

BASE

- Acronym contrived to be the opposite of ACID
- **B**asically **A**vailable,
- Soft state,
- Eventually Consistent
- Weak consistency stale data OK
- Availability first

BASE

- Acronym contrived to be the opposite of ACID
- **B**asically **A**vailable,
- Soft state,
- Eventually Consistent
- Weak consistency stale data OK
- Availability first
- Best effort. Optimistic Concurrency Control

BASE

- Acronym contrived to be the opposite of ACID
- **B**asically **A**vailable,
- Soft state,
- Eventually Consistent
- Weak consistency stale data OK
- Availability first
- Best effort. Optimistic Concurrency Control
- Approximate answers OK

BASE

- Acronym contrived to be the opposite of ACID
- **B**asically **A**vailable,
- Soft state,
- Eventually Consistent
- Weak consistency stale data OK
- Availability first
- Best effort. Optimistic Concurrency Control
- Approximate answers OK
- Simpler and faster

Advantages of NoSQL

- Cheap, easy to implement (open source)

Advantages of NoSQL

- Cheap, easy to implement (open source)
- Data Replication to multiple nodes

Advantages of NoSQL

- Cheap, easy to implement (open source)
- Data Replication to multiple nodes
- No single point of failure

Advantages of NoSQL

- Cheap, easy to implement (open source)
- Data Replication to multiple nodes
- No single point of failure
- Down nodes easily replaced

Advantages of NoSQL

- Cheap, easy to implement (open source)
- Data Replication to multiple nodes
- No single point of failure
- Down nodes easily replaced
- Can scale up and down

Advantages of NoSQL

- Cheap, easy to implement (open source)
- Data Replication to multiple nodes
- No single point of failure
- Down nodes easily replaced
- Can scale up and down
- No Schema

Disadvantages of NoSQL

- JOIN Queries - Restructure Data ?

Disadvantages of NoSQL

- JOIN Queries - Restructure Data ?
- Aggregate Queries - Pre-computation ?

Disadvantages of NoSQL

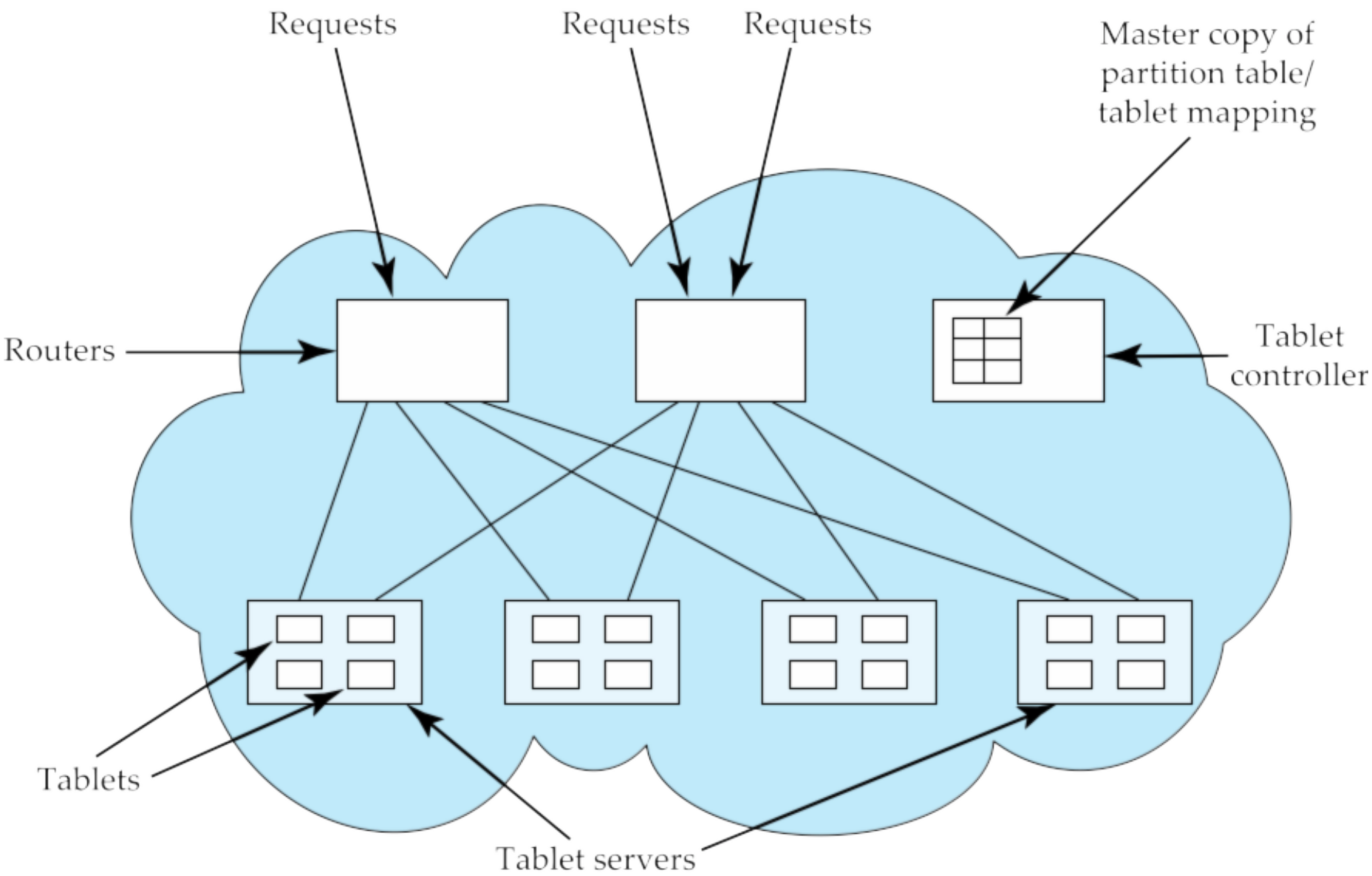
- JOIN Queries - Restructure Data ?
- Aggregate Queries - Pre-computation ?
- ACID Transaction - Available Partly.

Disadvantages of NoSQL

- JOIN Queries - Restructure Data ?
- Aggregate Queries - Pre-computation ?
- ACID Transaction - Available Partly.
- SQL is unavailable - ORM and other query languages

Disadvantages of NoSQL

- JOIN Queries - Restructure Data ?
- Aggregate Queries - Pre-computation ?
- ACID Transaction - Available Partly.
- SQL is unavailable - ORM and other query languages
- Change applications already written in SQL


Requests
Requests
Requests
Master copy of partition table/ tablet mapping
Routers
Tablet controller
Tablets
Tablet servers

Basic Definitions and alternatives

- Entity - A row or Document or JSON Object

Basic Definitions and alternatives

- Entity - A row or Document or JSON Object
- Entity Type - Table or collection or JSON Class

Basic Definitions and alternatives

- Entity - A row or Document or JSON Object
- Entity Type - Table or collection or JSON Class
- Key-Value - Cell or Value

Basic Definitions and alternatives

- Entity - A row or Document or JSON Object
- Entity Type - Table or collection or JSON Class
- Key-Value - Cell or Value
- Metadata - stores keys, time-stamp, id, etc

Basic Definitions and alternatives

- Entity - A row or Document or JSON Object
- Entity Type - Table or collection or JSON Class
- Key-Value - Cell or Value
- Metadata - stores keys, time-stamp, id, etc
- Indexing - Sorted list of keys and addresses

Basic Definitions and alternatives

- Entity - A row or Document or JSON Object
- Entity Type - Table or collection or JSON Class
- Key-Value - Cell or Value
- Metadata - stores keys, time-stamp, id, etc
- Indexing - Sorted list of keys and addresses
- Expando - Adding key-value to existing document

Basic Definitions and alternatives

- Entity - A row or Document or JSON Object
- Entity Type - Table or collection or JSON Class
- Key-Value - Cell or Value
- Metadata - stores keys, time-stamp, id, etc
- Indexing - Sorted list of keys and addresses
- Expando - Adding key-value to existing document
- Polymodel - Inheritance of Entity Type

Basic API

- get(key) – Extract the value given a key
- put(key, value) – Create or update the value given its key
- delete(key) – Remove the key and its associated value
- execute(key, operation, parameters) – Invoke an operation to the value (given its key)

Other Databases

- Domain Specific Databases

Other Databases

- Domain Specific Databases
- Relaxes further domain specific constraints

Other Databases

- Domain Specific Databases
- Relaxes further domain specific constraints
- Graph Databases - Titan, Neo4j

Other Databases

- Domain Specific Databases
- Relaxes further domain specific constraints
- Graph Databases - Titan, Neo4j
- Document Database - CouchDB, S3

Other Databases

- Domain Specific Databases
- Relaxes further domain specific constraints
- Graph Databases - Titan, Neo4j
- Document Database - CouchDB, S3
- Hash-Like Databases - Voldemort, Scalaris

Other Databases

- Domain Specific Databases
- Relaxes further domain specific constraints
- Graph Databases - Titan, Neo4j
- Document Database - CouchDB, S3
- Hash-Like Databases - Voldemort, Scalaris
- Key Value Systems like Lucene

Tips and Tricks

- First ensure you need NoSQL. Petabyte scale websites like Quora are still using MySQL
- High Read / Update frequency 500 writes / sec / GB RAM
- Log Analysis, Creating Warehouses, Backup, etc
- Prefer Cloud Services to avoid maintenance
- Index everything properly
- Entity update close to 1 write / sec

Questions ?

Questions ?

Acknowledgments

- Thank You IEEE
- Thank You Karthik Senthil
- Thank You for attending

Ashish Kedia - 12IT14