Principles and Techniques of DBMS 13 NoSQL DBMS

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*

Shanghai Jiao Tong University

Shanghai, China

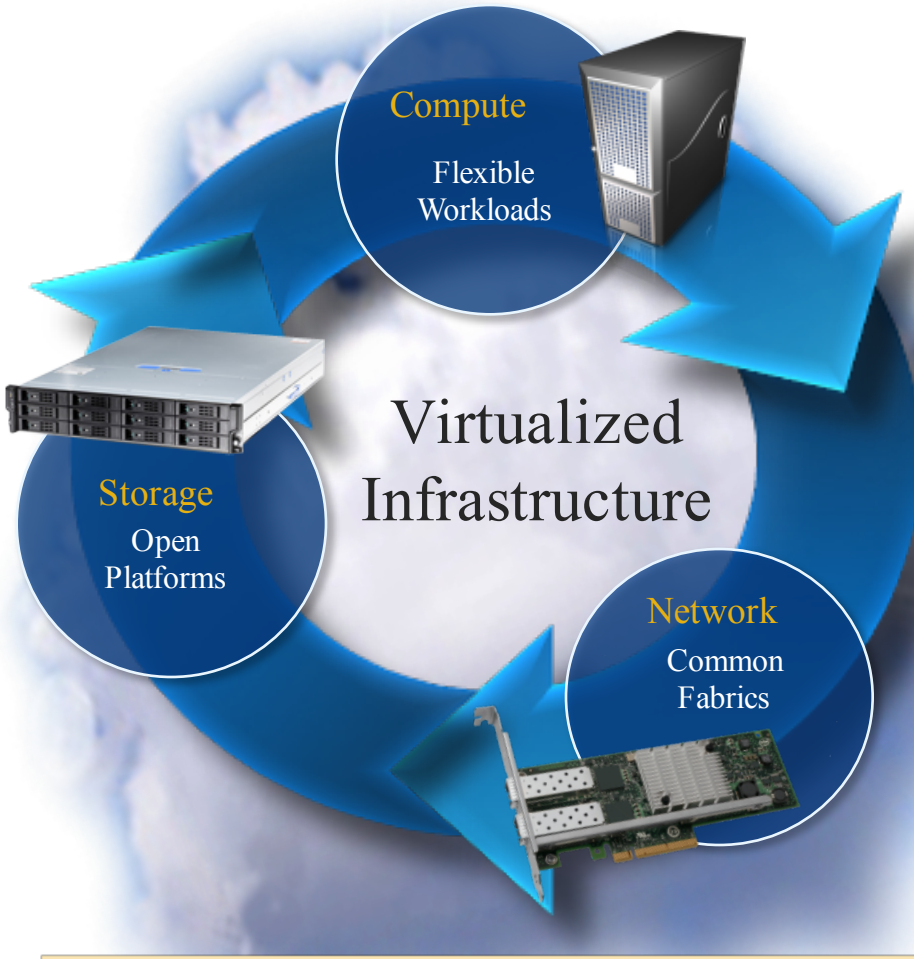
http://reins.se.sjtu.edu.cn/~chenhp

e-mail: chen-hp@sjtu.edu.cn

Overview

- Big Data !
  - A Grant Challenge to DBMS
- When RDBMS is standing in front of Big Data
  - You really want me to do it?!
- Why NoSQL?
  - High scalability!

Data Centers are Evolving

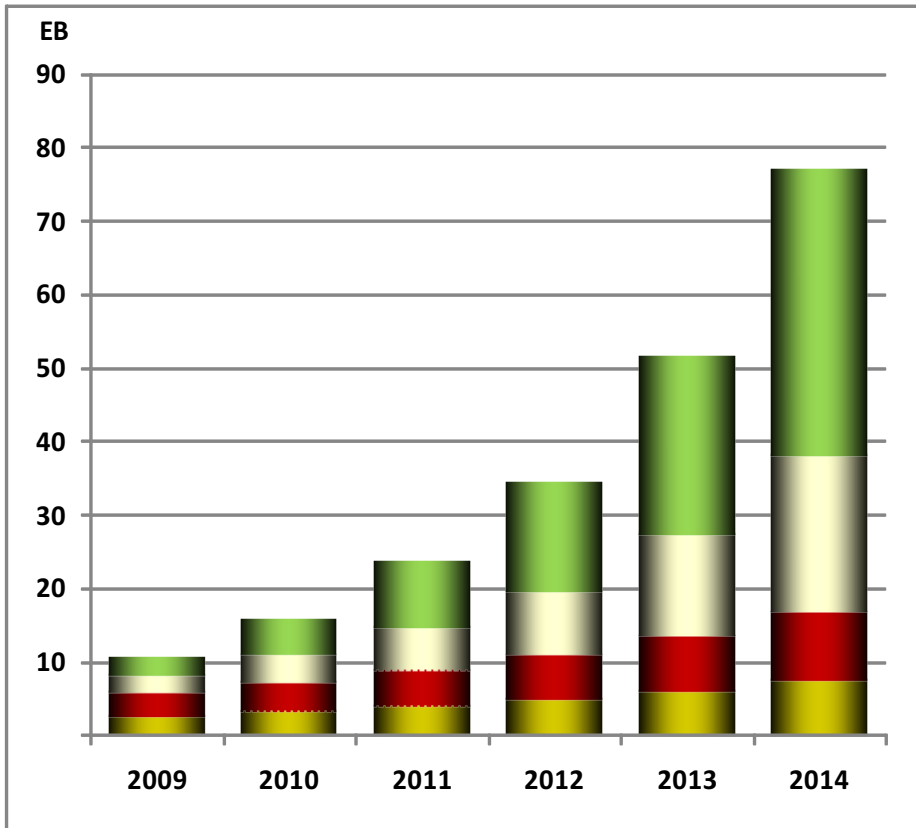


- Data centers are built upon **three fundamental pillars**:
  - **Compute**
  - **Storage**
  - **Networking**
- All three are critical for efficient data center operations
  - **Balanced** in performance and utilization

A Balanced Data Center is Essential for Efficiency

IDC Storage Capacity Growth

- **Structured Data (23.6%)**
  - Traditional Enterprise Database
- **Replicated Data (24.2%)**
  - Backups
  - Data warehouses
- **Unstructured Data (54.8%)**
  - Archives
- **Content Depots (75.6%)**
  - Web
  - Email
  - Document sharing
  - Social network content (pictures/videos)

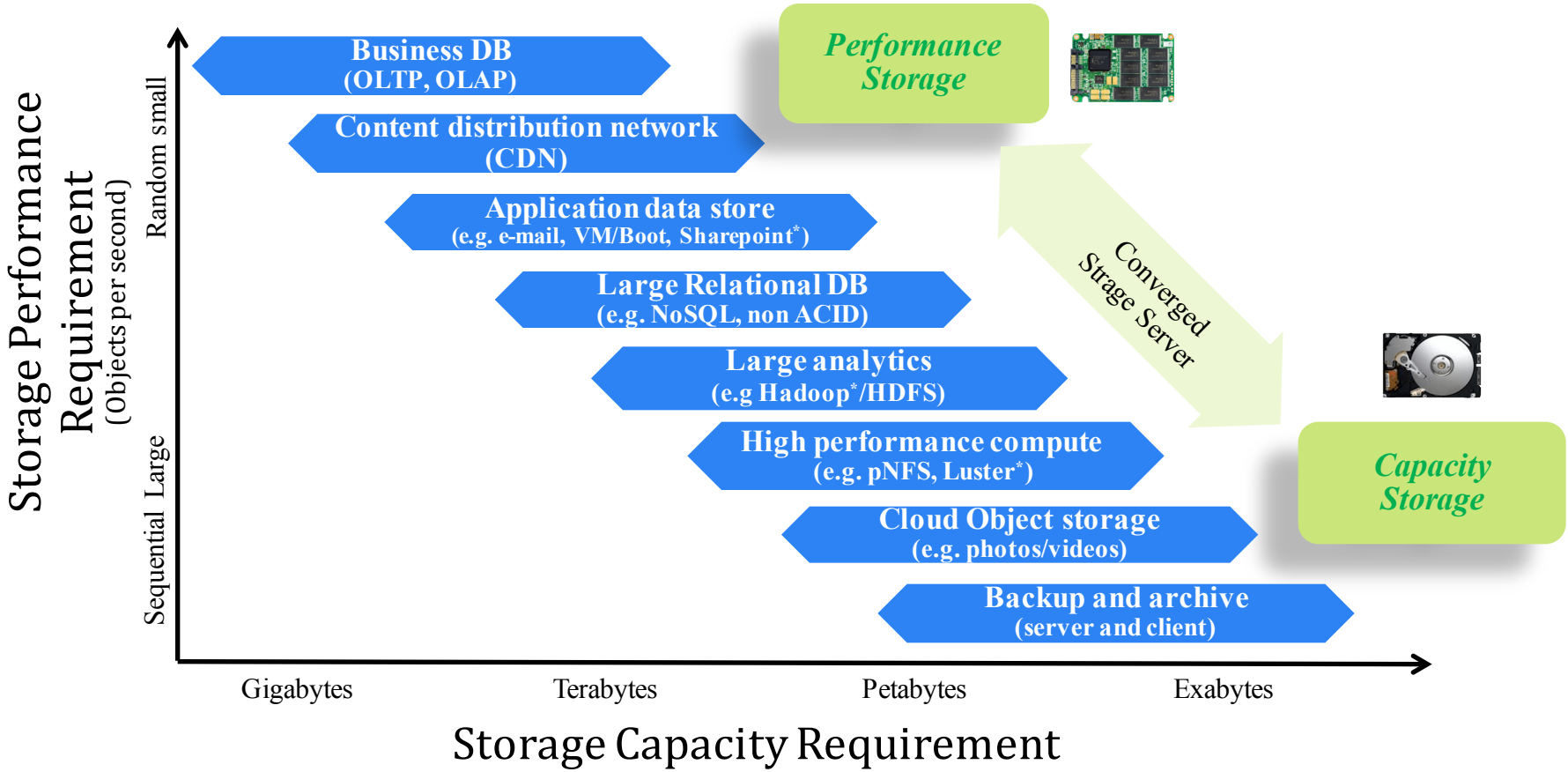


2012 Deployment Estimate:

~7.6 million drives
~500,000 storage systems‡

Usage Models Dictate the Solutions

REliable, INtelligent & Scalable Systems



Key Storage Usage Models Have Differing Requirements

Thus Need New Benchmarks

Big Data ? Really ?

- This flood of data is coming from many sources.
- Consider the following:
  - The New York Stock Exchange generates about one terabyte of new trade data per day.
  - Facebook hosts approximately 10 billion photos, taking up one petabyte of storage.
  - Ancestry.com, the genealogy site, stores around 2.5 petabytes of data.
  - The Internet Archive stores around 2 petabytes of data, and is growing at a rate of 20 terabytes per month.
  - The Large Hadron Collider near Geneva, Switzerland, will produce about 15 petabytes of data per year.

Big Data !

- More data usually beats better algorithms
- The good news is that
  - Big Data is here
- The bad news is that
  - we are struggling to store and analyze it

Data Storage and Analysis

- The problem is simple
  - The storage capacities of hard drives have increased massively
  - The rate at which data can be read from drives—have not kept up
  - One typical drive from 1990 could store 1,370 MB of data and had a transfer speed of 4.4 MB/s
  - At present, the transfer speed is around 100 MB/s, this is a long time to read all data on a single drive—and writing is even slower
- The obvious way to reduce the time is to read from multiple disks at once.

Data Storage and Analysis

- Only using one hundredth of a disk may seem wasteful.
  - But we can store one hundred datasets, each of which is one terabyte, and provide shared access to them.
  - We can imagine that the users of such a system would be happy to share access in return for shorter analysis times,
  - and, statistically, that their analysis jobs would be likely to be spread over time, so they wouldn't interfere with each other too much.

- There's more to being able to read and write data in parallel to or from multiple disks, though.

Data Storage and Analysis

- The first problem to solve is hardware failure
- As soon as you start using many pieces of hardware, the chance that one will fail is fairly high.
- A common way of avoiding data loss is through replication:
  - Redundant copies of the data are kept by the system so that in the event of failure, there is another copy available.
  - This is how RAID works.

Data Storage and Analysis

- The second problem is that most analysis tasks need to be able to combine the data in some way
  - data read from one disk may need to be combined with the data from any of the other 99 disks.

- Various distributed systems allow data to be combined from multiple sources, but doing this correctly is notoriously challenging.
  - MapReduce provides a programming model that abstracts the problem from disk reads and writes, transforming it into a computation over sets of keys and values.

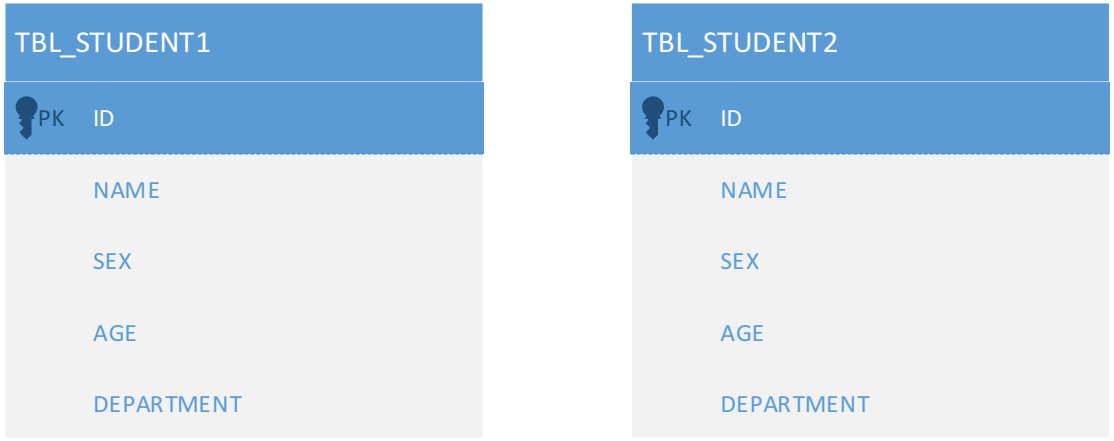
How about RDBMS ?

- What happens if we distribute the data of RDBMS into multiple physical machines?

- The problem(s) of single table
  - When a SQL statement is being executed, the table(s) will be locked optimistically/pessimistically in order to guarantee the integrity of data.
  - The shared lock allows other thread(s) to read but not write the table
  - The excluded lock denies any access from other thread(s), i.e. other statements will be queued and wait for the lock released.
  - For the latter, the performance is pretty poor if the number of statements is large.

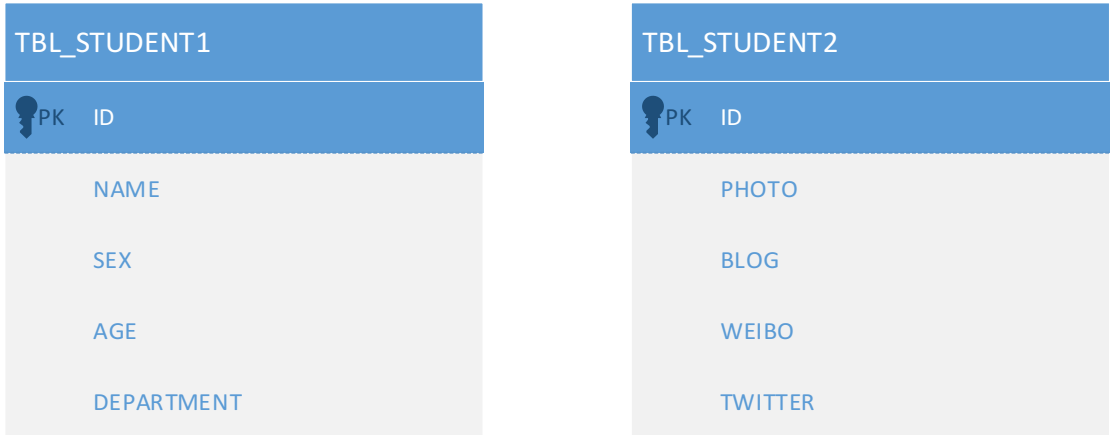
How about RDBMS ?

- To split tables horizontally
  - Store the data into several tables with same schemas in order to reduce the probability of access confliction.
  - For example, the TBL_STUDENT is split into two tables TBL_STUDENT1 and TBL_STUDENT2
  - The former holds all the freshmen and sophomores and the latter holds all the juniors and seniors
  - Now, we can query a freshmen “Zhang San” and a junior “Li Si” simultaneously.



How about RDBMS ?

- To split tables vertically
  - Store the data into several tables with complementary schemas in order to reduce the numbers of columns.
  - For example, the TBL_STUDENT is split into two tables TBL_STUDENT1 and TBL_STUDENT2
  - The former holds necessary information and the latter holds optional information
  - Now, we can query the basic information of a freshmen “Zhang San” in a table with fewer columns



How about RDBMS ?

- To do horizontal partitioning with table
  - Partition the data within a single table based on rows.
  - Rules
    - Range, Hash, Key, List and Composite

```
CREATE TABLE TBL_STUDENT
    ( ID int default NULL,
      NAME varchar(30) default NULL,
      BIRTHDAY date default NULL
    ) engine=myisam
  PARTITION BY RANGE (year(BIRTHDAY)) (PARTITION p0 VALUES LESS THAN (1995),
  PARTITION p1 VALUES LESS THAN (1996) , PARTITION p2 VALUES LESS THAN (1997) ,
  PARTITION p3 VALUES LESS THAN (1998) , PARTITION p4 VALUES LESS THAN (1999) ,
  PARTITION p5 VALUES LESS THAN (2000) , PARTITION p6 VALUES LESS THAN (2001) ,
  PARTITION p7 VALUES LESS THAN (2002) , PARTITION p8 VALUES LESS THAN (2003) ,
  PARTITION p9 VALUES LESS THAN (2004) , PARTITION p10 VALUES LESS THAN (2010),
  PARTITION p11 VALUES LESS THAN MAXVALUE );
```

How about RDBMS ?

- To do vertical partitioning with table
  - Partition the data within a single table based on columns.
  - Needs to be implemented manually
  - But it can improve the performance significantly

How about RDBMS ?

- Why partitioning?

- Another trend in disk drives
  - Seek time is improving more slowly than transfer rate.

- If the data access pattern is dominated by seeks
  - it will take longer to read or write large portions of the dataset than streaming through it, which operates at the transfer rate.

How about RDBMS ?

- Why partitioning?

- On the other hand, for updating a small proportion of records in a database
  - A traditional B-Tree (the data structure used in relational databases, which is limited by the rate it can perform seeks) works well.
  - For updating the majority of a database, a B-Tree is less efficient.

ORM for Shards

- Hibernate Shards
  - The primary goal of Hibernate Shards is to enable application developers to query and transact against sharded datasets using the standard Hibernate Core API.

```
CREATE TABLE WEATHER_REPORT (
    REPORT_ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    CONTINENT ENUM('AFRICA', 'ANTARCTICA', 'ASIA', 'AUSTRALIA', 'EUROPE',
    'NORTH AMERICA', 'SOUTH AMERICA'),
    LATITUDE FLOAT,
    LONGITUDE FLOAT,
    TEMPERATURE INT,
    REPORT_TIME TIMESTAMP
);
```

ORM for Shards

- Hibernate Shards

```
public class WeatherReport {
    private Integer reportId;
    private String continent;
    private BigDecimal latitude;
    private BigDecimal longitude;
    private int temperature;
    private Date reportTime;
    ... // getters and setters
}
```

ORM for Shards

- Hibernate Shards

```
<hibernate-mapping package="org.hibernate.shards.example.model">
    <class name="WeatherReport" table="WEATHER_REPORT">
        <id name="reportId" column="REPORT_ID">
          <generator class="native"/>
        </id>
        <property name="continent" column="CONTINENT"/>
        <property name="latitude" column="LATITUDE"/>
        <property name="longitude" column="LONGITUDE"/>
        <property name="temperature" column="TEMPERATURE"/>
        <property name="reportTime" type="timestamp" column="REPORT_TIME"/>
    </class>
</hibernate-mapping>
```

ORM for Shards

- Hibernate Shards

```
public SessionFactory createSessionFactory() {
  Configuration config = new Configuration();
  config.configure("weather.hibernate.cfg.xml");
  config.addResource("weather.hbm.xml");
  return config.buildSessionFactory();
}
```

```
<hibernate-configuration>
  <session-factory name="HibernateSessionFactory">
    <property name="dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql://localhost:3306/mydb</property>
    <property name="connection.username">my_user</property>
    <property name="connection.password">my_password</property>
  </session-factory>
</hibernate-configuration>
```

ORM for Shards

- Hibernate Shards

```
public SessionFactory createSessionFactory() {
 Configuration prototypeConfig = new
                                    Configuration().configure("shard0.hibernate.cfg.xml");
  prototypeConfig.addResource("weather.hbm.xml");
  List<ShardConfiguration> shardConfigs = new ArrayList<ShardConfiguration>();
  shardConfigs.add(buildShardConfig("shard0.hibernate.cfg.xml"));
  shardConfigs.add(buildShardConfig("shard1.hibernate.cfg.xml"));
  shardConfigs.add(buildShardConfig("shard2.hibernate.cfg.xml"));

  ShardStrategyFactory shardStrategyFactory = buildShardStrategyFactory();
  ShardedConfiguration shardedConfig = new ShardedConfiguration(
    prototypeConfig,
    shardConfigs,
    shardStrategyFactory);
    return shardedConfig.buildShardedSessionFactory();
 }
```

ORM for Shards

- Hibernate Shards

```
ShardStrategyFactory buildShardStrategyFactory() {
  ShardStrategyFactory shardStrategyFactory = new ShardStrategyFactory() {
    public ShardStrategy newShardStrategy(List<ShardId> shardIds) {
      RoundRobinShardLoadBalancer loadBalancer = new
                          RoundRobinShardLoadBalancer(shardIds);
      ShardSelectionStrategy pss = new
                          RoundRobinShardSelectionStrategy(loadBalancer);
      ShardResolutionStrategy prs = new AllShardsShardResolutionStrategy(shardIds);
      ShardAccessStrategy pas = new SequentialShardAccessStrategy();
      return new ShardStrategyImpl(pss, prs, pas);
    }
  };
  return shardStrategyFactory;
}
```

ORM for Shards

- Hibernate Shards

```
ShardConfiguration buildShardConfig(String configFile) {
  Configuration config = new Configuration().configure(configFile);
  return new ConfigurationToShardConfigurationAdapter(config);
}
```

ORM for Shards

- Hibernate Shards

```
<!-- Contents of shard0.hibernate.cfg.xml -->
  <hibernate-configuration>
    <session-factory name="HibernateSessionFactory0"> <!-- note the different name -->
      <property name="dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
      <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
      <property name="connection.url">jdbc:mysql://dbhost0:3306/mydb</property>
      <property name="connection.username">my_user</property>
      <property name="connection.password">my_password</property>
      <property name="hibernate.connection.shard_id">0</property> <!-- new -->
      <property name="hibernate.shard.enable_cross_shard_relationship_checks">
          true
      </property>
    </session-factory>
</hibernate-configuration>
```

ORM for Shards

- Hibernate Shards

```
<!-- Contents of shard1.hibernate.cfg.xml -->
  <hibernate-configuration>
    <session-factory name="HibernateSessionFactory1"> <!-- note the different name -->
      <property name="dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
      <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
      <property name="connection.url">jdbc:mysql://dbhost0:3306/mydb</property>
      <property name="connection.username">my_user</property>
      <property name="connection.password">my_password</property>
      <property name="hibernate.connection.shard_id">1</property> <!-- new -->
      <property name="hibernate.shard.enable_cross_shard_relationship_checks">
          true
      </property>
    </session-factory>
</hibernate-configuration>
```

ORM for Shards

- Hibernate Shards

```
<hibernate-mapping package="org.hibernate.shards.example.model">
    <class name="WeatherReport" table="WEATHER_REPORT">
        <id name="reportId" column="REPORT_ID" type="long">
            <generator class="org.hibernate.shards.id.ShardedTableHiLoGenerator"/>
        </id>
        <property name="continent" column="CONTINENT"/>
        <property name="latitude" column="LATITUDE"/>
        <property name="longitude" column="LONGITUDE"/>
        <property name="temperature" column="TEMPERATURE"/>
        <property name="reportTime" type="timestamp" column="REPORT_TIME"/>
    </class>
</hibernate-mapping>
```

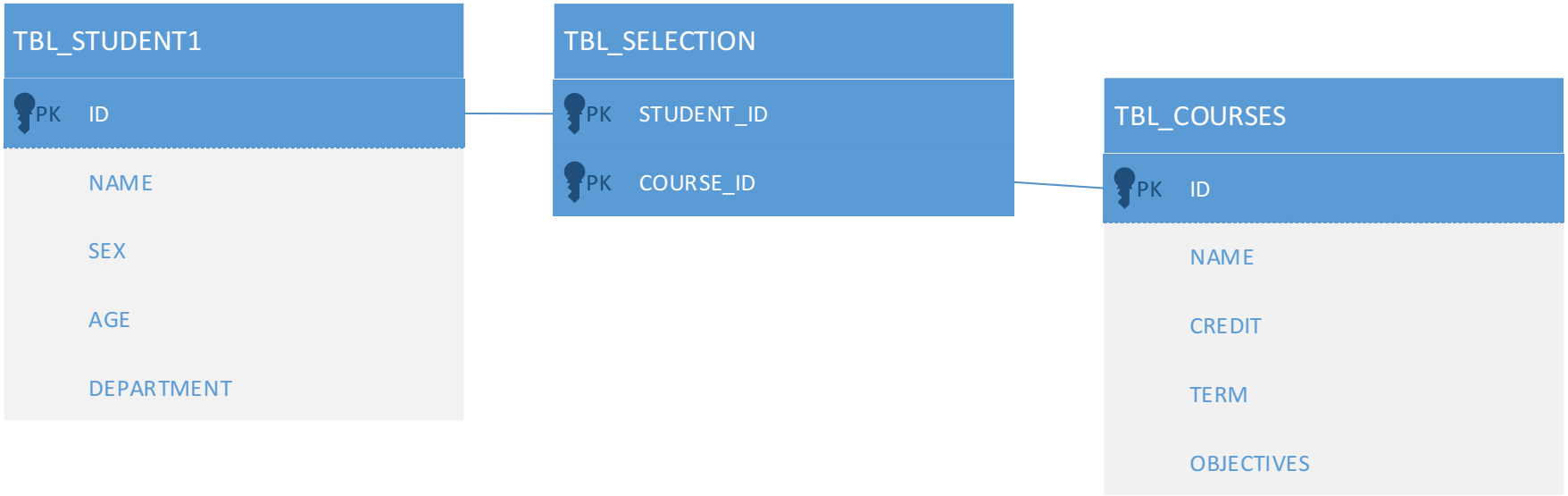
ORM for Shards

- Hibernate Shards

```
@Entity
@Table(name="WEATHER_REPORT")
public class WeatherReport {
        @Id @GeneratedValue(generator="WeatherReportIdGenerator")
        @GenericGenerator(name="WeatherReportIdGenerator",
                                strategy="org.hibernate.shards.id.ShardedUUIDGenerator")
        @Column(name="REPORT_ID")
        private Integer reportId;
        @Column(name="CONTINENT")
        private String continent;
        @Column(name="LATITUDE")
        private BigDecimal latitude;
        @Column(name="LONGITUDE")
        private BigDecimal longitude;
        @Column(name="TEMPERATURE")
        private int temperature;
        @Column(name="REPORT_TIME")
        private Date reportTime;
        ... // getters and setters
}
```

How about RDBMS ?

- Even if we can use partitioning or splitting, what can we with relationships between tables?
  - It is hard to be sharded.



- How to deal with the semi-structured and unstructured massive data with RDBMS?

How about RDBMS ?

- *Structured data*
  - is data that is organized into entities that have a defined format, such as XML documents or database tables that conform to a particular predefined schema.
  - This is the realm of the RDBMS.
- *Semi-structured data*
  - on the other hand, is looser, and though there may be a schema, it is often ignored, so it may be used only as a guide to the structure of the data
  - for example, a spreadsheet, in which the structure is the grid of cells, although the cells themselves may hold any form of data.
- *Unstructured data*
  - does not have any particular internal structure
  - for example, plain text or image data.

NoSQL DBMS

- Since RDBMS is incompetent for massive data storage and processing
  - NoSQL DBMS has become an emerging technology as a complement to an RDBMS
  - For example, MapReduce

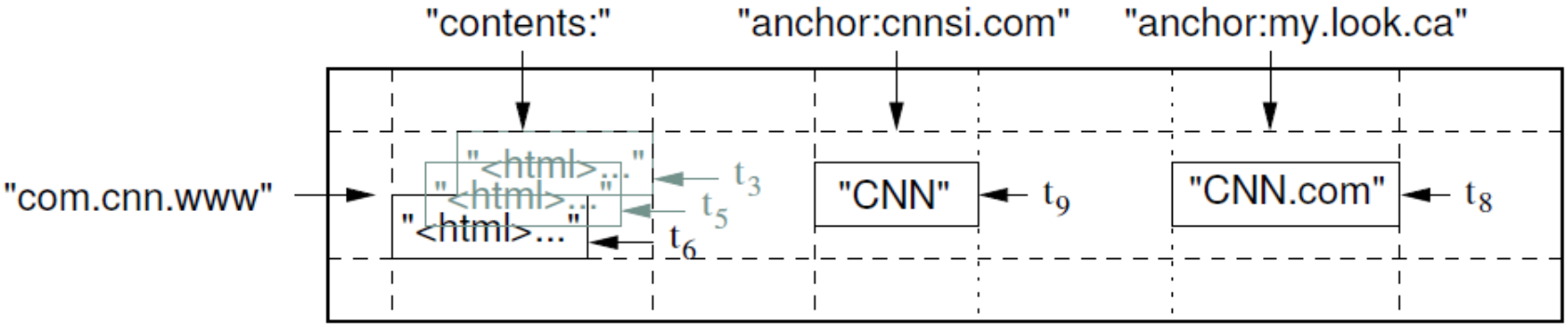
	Traditional RDBMS	MapReduce
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and write many times	Write once, read many times
Structure	Static schema	Dynamic schema
Integrity	High	Low
Scaling	Nonlinear	Linear

No-SQL DBMS

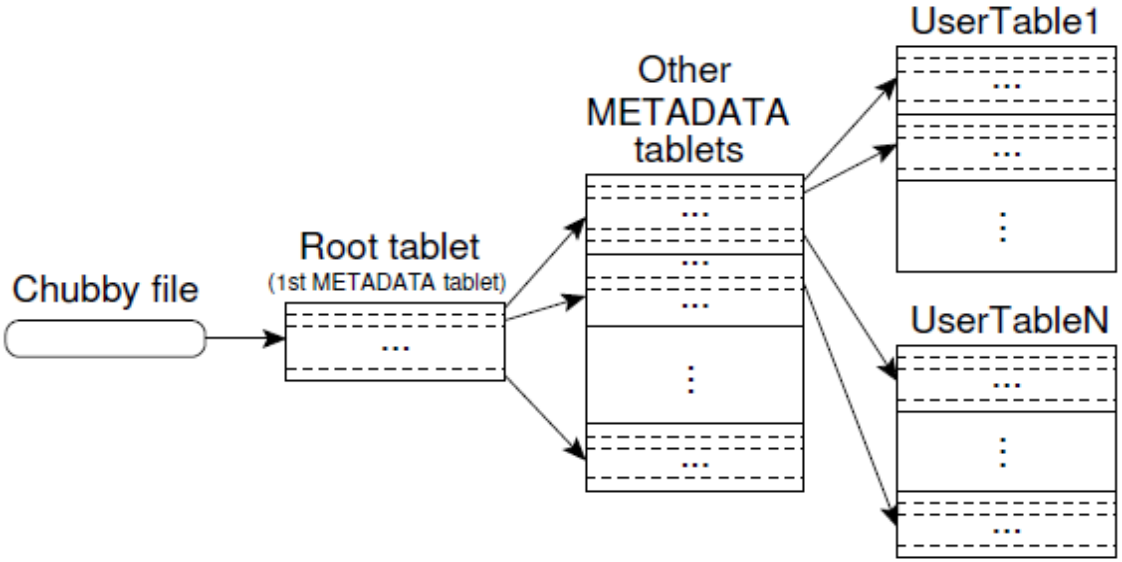
- Bigtable: A Distributed Storage System for Structured Data
  - ACM Transactions on Computer Systems, 2008, 26:1–26.
  - http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/zh-CN//archive/bigtable-osdi06.pdf
- Dynamo: amazon's highly available key-value store
  - Symposium on Operating Systems Principles, 2007:205–220.
  - http://web.archive.org/web/20120129154946/http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf
- Cassandra
  - http://cassandra.apache.org/
- MemcacheDB
  - http://memcachedb.org/
- Apache CouchDB
  - http://couchdb.apache.org/
- MongoDB
  - http://www.mongodb.org/

- A Bigtable is a sparse, distributed, persistent multidimensional sorted map.
  - The map is indexed by a row key, column key, and a timestamp;
  - each value in the map is an uninterpreted array of bytes.

(row:string, column:string, time:int64) -> string

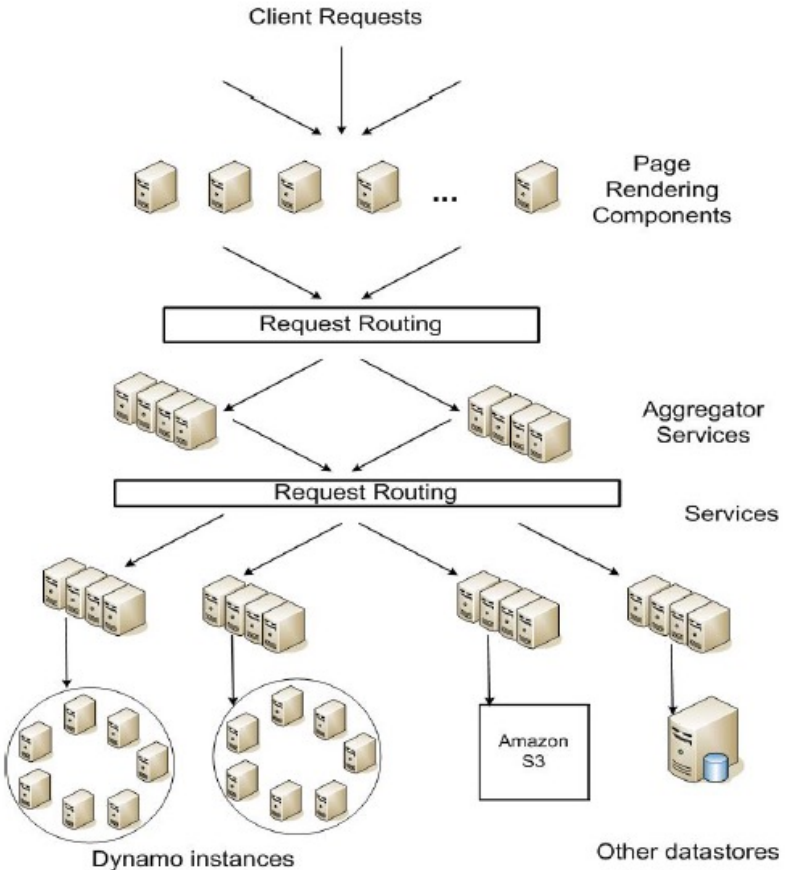


- A Bigtable cluster stores a number of tables.
  - Each table consists of a set of tablets
  - and each tablet contains all data associated with a row range.
  - Initially, each table consists of just one tablet.
  - As a table grows, it is automatically split into multiple tablets, each approximately 100-200 MB in size by default.

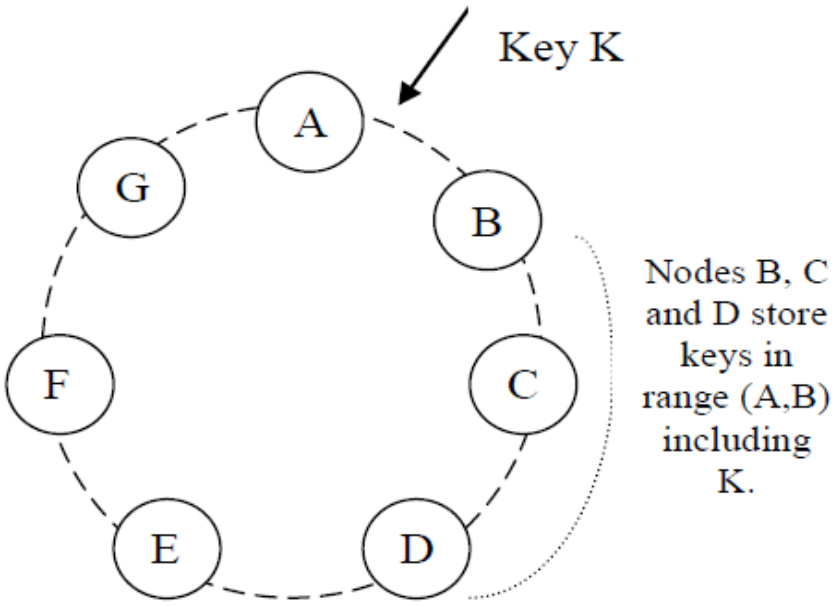


Dynamo

- Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience.



- Replication
  - In this figure, node B replicates the key *k* at nodes C and D in addition to storing it locally. Node D will store the keys that fall in the ranges (A, B], (B, C], and (C, D].



Cassandra

- Cassandra is a highly scalable, eventually consistent, distributed, structured key-value store.

- Cassandra brings together the distributed systems technologies from Dynamo and the data model from Google's BigTable.
  - Like Dynamo, Cassandra is eventually consistent.
  - Like BigTable, Cassandra provides a ColumnFamily-based data model richer than typical key/value systems.

MemcacheDB

- MemcacheDB is a distributed key-value storage system designed for persistent.

- It is **NOT** a cache solution, but a persistent storage engine for fast and reliable key-value based object storage and retrieval.
  - It conforms to memcache protocol(not completed), so any memcached client can have connectivity with it.
  - MemcacheDB uses Berkeley DB as a storing backend, so lots of features including transaction and replication are supported.

CouchDB

- Apache CouchDB™ is a database that uses JSON for documents, JavaScript for MapReduce queries, and regular HTTP for an API

- CouchDB is often categorized as a "NoSQL" database
  - a term that became increasingly popular in late 2009, and early 2010.
  - While this term is a rather generic characterization of a database, or data store, it does clearly define a break from traditional SQL-based databases.

- A CouchDB database lacks a schema, or rigid pre-defined data structures such as tables.
  - Data stored in CouchDB is a JSON document(s).
  - The structure of the data, or document(s), can change dynamically to accommodate evolving needs.

MongoDB

- MongoDB (from "**humongous**") is an open source document database, and the leading NoSQL database. Written in C++, MongoDB features:
  - Document-Oriented Storage
  - Full Index Support
  - Replication & High Availability
  - Auto-Sharding
  - Querying
  - Fast In-Place Updates
  - Map/Reduce
  - GridFS
  - Commercial Support

Project

- Requirement
  - Using MongoDB, the user profiles are stored into NoSQL database.
  - The details of user profile could consist of photos, videos and rich text.

Reference

- Hadoop: The Definitive Guide, 2nd edition
  - Tom White, O'Reilly/Yahoo Press
- Hibernate Shards-Horizontal Partitioning With Hibernate,
  - http://www.hibernate.org/subprojects/shards/docs

Thank You!