# Big Data Storage

## ▲Introduction to Big Data

Big data is a broad term for data sets and it can be described by the following main 3Vs and other 2Vs characteristics:

Volume [容积] (huge large amount of data: terabytes, petabytes, exabytes) [数据集的大容量]

Velocity [速率] (speed of data in and out: real-time, streaming) [数据出入的速度]

Variety [多样性] (range of data types and sources, non-relational data such as nested relation, documents, XML data, web data, graph, multimedia, flexible schema or no schema) [数据类型和来源的范围]

Two more Vs

Veracity [准确] (correctness and accuracy of information: data quality and reliability) [信息的正确性]

Value [价值] (use machine learning, data mining, statistics, visualization, decision analysis techniques to extract/mine/derive previously unknown insights from data and become actionable knowledge, business value) [有用作分析的价值]

Database Models:
File system
Hierarchical Model (IMS)
Network Model (IDMS)
Relational Model
Nested Relational Model
Entity-Relationship Approach
Object-Oriented (OO) Data Model
Deductive and Object-Oriented (DOOD)
Object Relational Data Model
Semi-structured Data Model (XML)
RDF and Linked Data
...

## ▲Issues and performance problems in RDBMS

•传统的关系型数据库(RDBMS)使用 SQL，不适合处理大量的数据

•关系模型中的正常形式是去除冗余并减少更新异常，但在物理数据库设计中添加冗余不会引起更新异常，相反可能会显著提高性能避免 join， join 操作是非常消耗算力的，故关系型模型可能不适合一些程序

## ▲NoSQL and categories

**NoSQL (not-only SQL)**

Flexible schema or no schema, avoidance of unneeded complexity which are designed to store data structures that are either simple or more similar to the ones of object-oriented programming languages compared to relational data structures. NoSQL softens the ACID properties in relational databases to allow horizontal scalability ().

Benefits: massive scalability & high throughput & higher performance & availability & quicker/cheaper to set up …

## ▲Consistency

Strong consistency: after the update completes, any subsequent access will return the updated value.

Weak consistency: a number of conditions need to be met before the updated value will be returned.

Eventual consistency: a consistency model used in distributed computing to achieve high availability that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.

**▲4 major categories for NoSQL databases:**

**•Key-value Stores**

•Key-value storage systems store large numbers (billions or even more) of small (KB-MB) sized records

•Records are partitioned across multiple machines

•Queries are routed by the system to appropriate machines

•Records are also replicated across multiple machines, to ensure availability even if a machine fails

(Key-value stores ensure that updates are applied to all replicas, to ensure that their values are consistent)

•A key-value store is like associate array:

data is represented in the form of array["key"] = value or hash table in main memory.

•Each data/object is stored, indexed, and accessed using a key value to access the hash table or array.

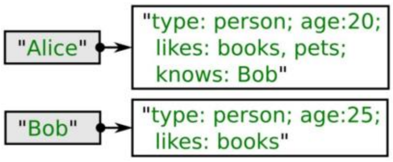
•Value is a single opaque collection of objects or data items

can be structured, semi-structured, or unstructured. It is just an un-interpreted string of bytes of arbitrary length.

•The meaning of the value in a key-value pair has to be interpreted by programmers.

•No concept of "foreign key", no join

data can be horizontally partitioned and distributed




Key-value stores may store:

Un-interpreted bytes, with an associated key [带有关联键的未解释字节]

E.g., Amazon S3, Amazon Dynamo

Wide-table (can have arbitrarily many attribute names) with associated key [带有关联键的宽表]

Google BigTable, Apache Cassandra, Apache Hbase, Amazon DynamoDB

Allows some operations (e.g., filtering) to execute on storage node

JSON [JavaScript 对象表示法] (lecture courseware 11a P19 example)

MongoDB, CouchDB (document model)

Typical operations include (but no modification):

INSERT new Key-Value pairs (or put)

LOOKUP value for a specified key (or get)

DELETE key and the value associated with it

Some systems also support range queries on key values

Key value stores are not full database systems

•Have no/limited support for transactional updates

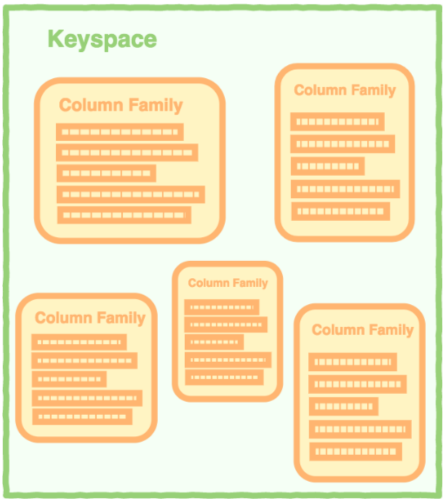
•Applications must manage query processing on their own

Not supporting above features makes it easier to build scalable data storage systems, also called NoSQL systems

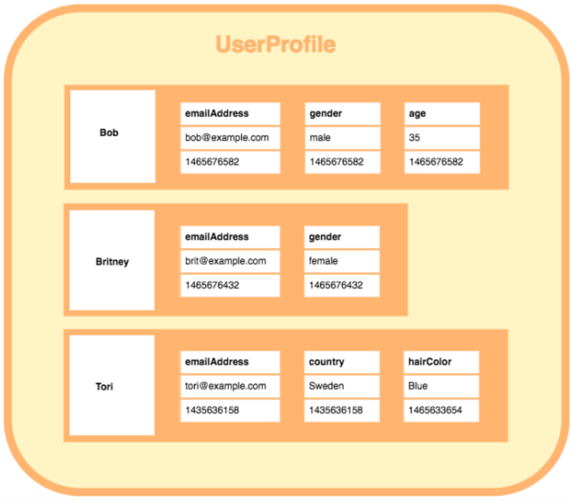
**•Wide-Column Stores**

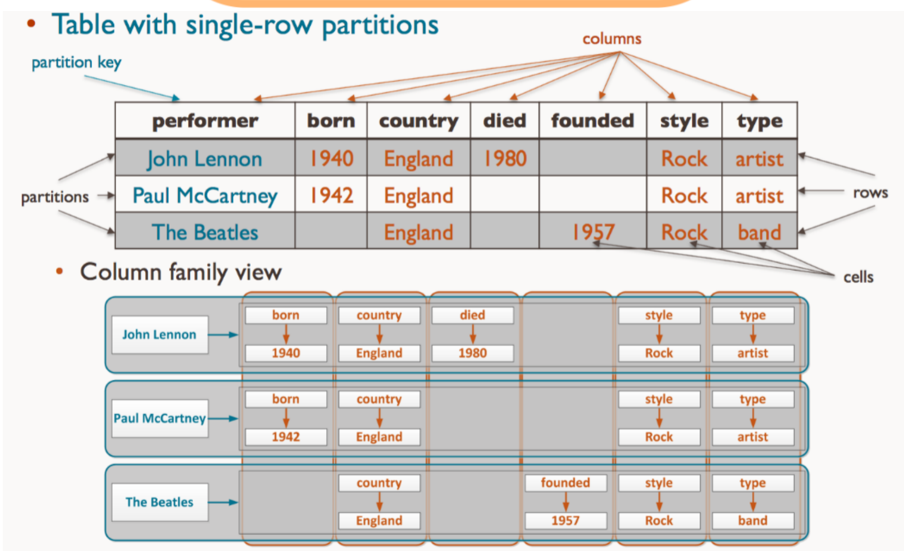
•Data is stored as tables. A table has a row-key and a pre-defined set of column-family columns.

•Each row in the table is uniquely identified by a row-key value.
•Each column family has a large & flexible number of columns (i.e. the No of columns may change from row to row) and each column has a name together with one or more values.
•A column-oriented DBMS stores data tables as column families of data rather than as rows of data (better for data compression).
•A keyspace in a wide-column store contains all the column families (like tables in the relational model), which contain rows, which contain columns (reference: [What is a Column Store Database?](#)):




•A column family containing 3 rows. Each row contains its own set of columns:




单行分区表(reference: [The Main NoSQL Database Types | Studio 3T](#))

●BigTable: Storing Web Pages

A sparse, distributed, persistent multi-dimensional sorted map.

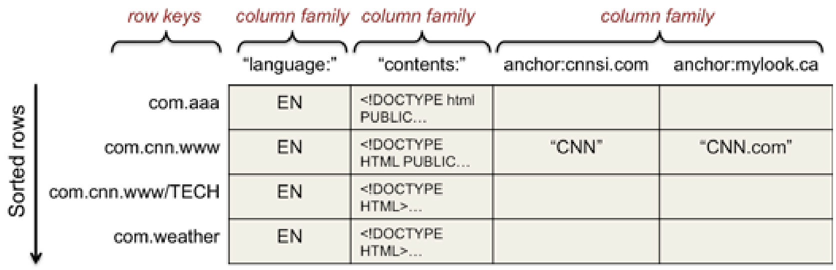
Used by several Google applications such as web indexing, MapReduce, Google Maps, Google Earth, YouTube, Gmail, etc.

The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

For webpage, the row-key value is a reversed url.

BigTable maintains data in lexicographic order by row key.

So webpages in the same domain are grouped together into contiguous rows.




Reference: [BigTable (rutgers.edu)]

●Document Stores

Schema languages are not powerful to express Object- Relationship-Attribute semantics in ER model.

Data is stored in so-called documents. (Arbitrary data in some (semi-)structured format: JSON, BSON, XML)

Data format is typically fixed, but the structure is flexible. (In a JSON-based document store, documents with completely different sets of attributes can be stored together)

●Graph Database

Best suited for representing data with a large number of interconnections

- especially when information about those interconnections is at least as important as the represented data
- for example, social relations or geographic data.

Graph databases allow for queries on the graph structure, e.g., relations between nodes or shortest paths.

Examples:

RDF(Resource Description Framework) graph and linked data & Google knowledge graph

## ▲SQL vs NoSQL

| Characteristics | SQL | NoSQL |
|---|---|---|
| Schema | Yes. Schema must be predefined and fixed. Schema evolution is difficult. | Schema is optional, may not be predefined. Schema can be semi structured or unstructured |
| Data type | Flat relations. Fixed length field/record for each relation defined. | Tree/graph structured data. Variable length, multi-valued attribute (repeating, nested tree), can add new tag names any time. |
| Data persistence | Databases are stored on disk drive, data persistence. Very slow to access as compared to in-memory stores. | In-memory, use pointer and hashing. Very fast to access. Need to convert data in memory from/to disk drive to archive data persistence. |

| | | |
|---|---|---|
| OLTP or OLAP | OLTP, mission critical online transaction applications. Only keep current database state. If historical data is required then need to use temporal database with time period to store. | OLAP for data warehouse and data analytics. Keep the historical data, time/date dimension is a must for meaningful data analysis. (Seldom mention time attribute in key-value store and data graph.) |
| Language to access the data | Standard DBMS declarative query language SQL. Operate on a set of tuples at a time basis. | Different imperative programming languages. Write programs (e.g. MapReduce, JSON programs with API's). |
| Update to data | Frequent update to database (transaction) | mainly have new data, no or seldom updates (deletion and addition) |
| Query optimization | Query optimizer of RDBMS | Optimization done by programmers for each of their programs. |
| DBMS | RDBMS | Not really, just as data stores |
| Answers for queries | Return accurate/precise query answers | Return "best guess" or "an opinion" answers – similar to data mining and IR answers |
| ACID | Yes, consistency is the most important issue for OLTP applications | Emphasis on speed performance, use eventual consistent. If no updates, then ACID is not required. |
| Join operation | Yes, queries may involve many joins, can be very slow. | Avoid or no join. Use redundant data to speed up processing |
| Ad hoc user queries | Write SQL programs or RDB keyword query search | Need programmers to write programs. |
| Distributed & parallel processing | Limited | Yes. Data can be partitioned horizontally and/or vertically and distributed to nodes, and process the partitioned data in parallel. Efficient for such applications. |

**🔺Big data storage**

**🔺Big Data Storage Systems:**

- •Distributed file systems
- •Shading across multiple databases
- •Key-value storage systems (all the NoSQL storage systems as key-value stores)
- •Parallel and distributed databases

**🔺MapReduce [分布式计算系统]**

MapReduce 详解_burpee 的博客-CSDN 博客_mapreduce

**🔺Distributed File Systems:**

•A distributed file system stores data across a large collection of machines, but provides single file system view

•Highly scalable distributed file system for large data-intensive applications. (e.g. 10K nodes, 100 million files, 10 PB)

•Provides redundant storage of massive amount of data on cheap and unreliable computers

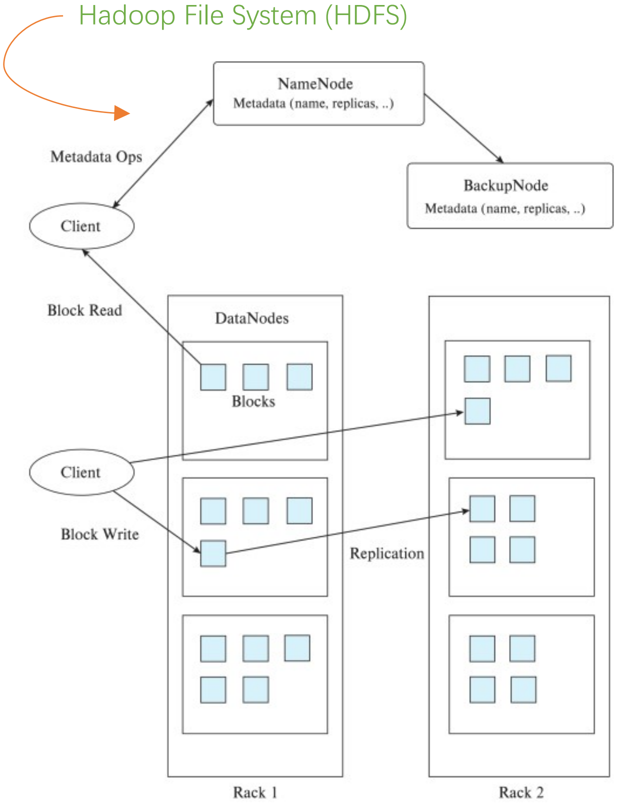
(1) Files are replicated to handle hardware failure

(2) Detect failures and recovers from them

Examples:

Google File System (GFS)

Hadoop File System (HDFS)




•Single Namespace for entire cluster

•Files are broken up into blocks (typically 64 MB block size and each block replicated on multiple DataNodes)

•Client [客户] can find location of blocks from NameNode and accesses data directly from DataNode

(System architecture)

NameNode:

Maps a filename to list of Block IDs

Maps each Block ID to DataNodes containing a replica of the block

DataNode:

Maps a Block ID to a physical location on disk

Data Coherency:

Write-once-read-many access model

Client can only append to existing files

Distributed file systems are good for millions of large files, but have very high overheads and poor performance with billions of smaller tuples

Sharding: partition data across multiple databases

优点：伸缩性好，易于实现

缺点：由于跨越了数个数据库，所以数据量越大，失败的机率也会越大

## ▲Parallel [并行] Databases and Data Stores

•Parallel databases run multiple machines (cluster)

•Parallel databases were designed for smaller scale (10s to 100s of machines)

•Replication used to ensure data availability despite machine failure

Supporting scalable data access

- Approach 1: Memcached or other caching mechanisms at application servers, to reduce database access Limited in scalability
- Approach 2: Partition ("shard") data across multiple separate database servers
- Approach 3: Use existing parallel databases
- Approach 4: Massively Parallel Key-Value Data Store

Other is sharding systems and key-value stores don't support many relational features, such as joins, integrity constraints, etc., across partitions.

### ▲The MapReduce Paradigm

Platform for reliable, scalable parallel computing

Abstracts issues of distributed and parallel environment from programmer map() reduce()

Paradigm dates back many decades

Data storage/access typically done using distributed file systems or key-value stores

Some specific examples: lecture courseware 11a P41

- 
- 
- 

### ▲MapReduce vs database

MapReduce is widely used for parallel processing

- Google, Yahoo, and 100's of other companies
- Example uses: compute PageRank, build keyword indices, do data analysis of web click logs, ….
- Allows procedural code in map and reduce functions
- Allows data of any type

Many real-world uses of MapReduce cannot be expressed in SQL

But many computations are much easier to express in SQL because MapReduce is cumbersome for writing simple queries

Relational operations (select, project, join, aggregation, etc.) can be expressed using MapReduce

SQL queries can be translated into MapReduce infrastructure for execution

- Apache Hive SQL, Apache Pig Latin, Microsoft SCOPE