第十四章 大数据管理

目录

- 大数据概述
- 大数据应用
- 大数据管理系统介绍
  - NOSQL
  - NEWSQL

数据模型的发展

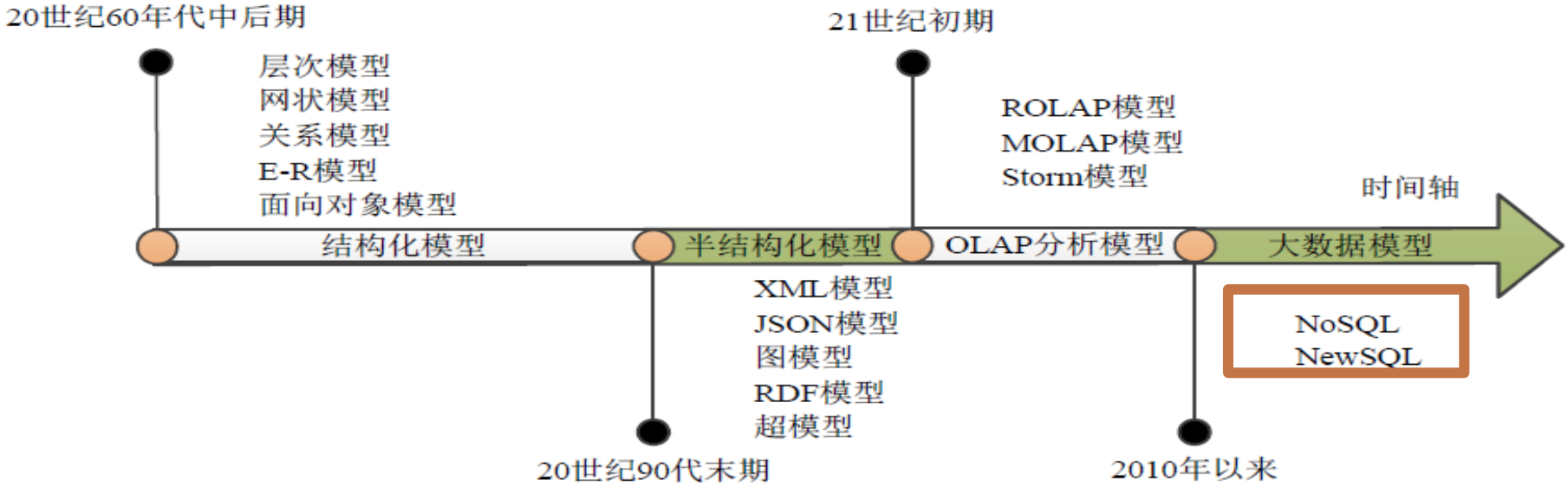


Fig.1 Timeline for the development of data models

图 1 数据模型的发展时间轴

大数据的来源

概念的发展：

- 超大规模数据（20世纪70年代，数百万条）
- 海量数据（21世纪，更大，更丰富的数据集）
- 大数据（2008年9月science发表的big data: science in the Petabyte era）

大数据定义

* 指无法在可容忍的时间里用现有的IT技术和软硬件工具对其进行感知，获取，管理，处理和服务的数据集合。
* 通常被认为是PB（1000TB）或EB或更高数据量级的数据，包括结构化的，半结构化的和非结构化的数据。

I byte=8 bit
1KB=1024 byte
1MB=1024 KB
1G=1024MB
1TB=1024GB
1PB=1024TB
1EB=1024PB

大数据的特点

- 巨量：人均5.2TB
- 多样：文本，图像，图形，音频，视频，博客等
- 快变：实时性，动态，快速产生
- 价值：潜在，巨大。
- Volume,Variety,Velocity,Variability,Veracity,
- Complexity,Value

大数据应用

* 互联网文本大数据管理与挖掘

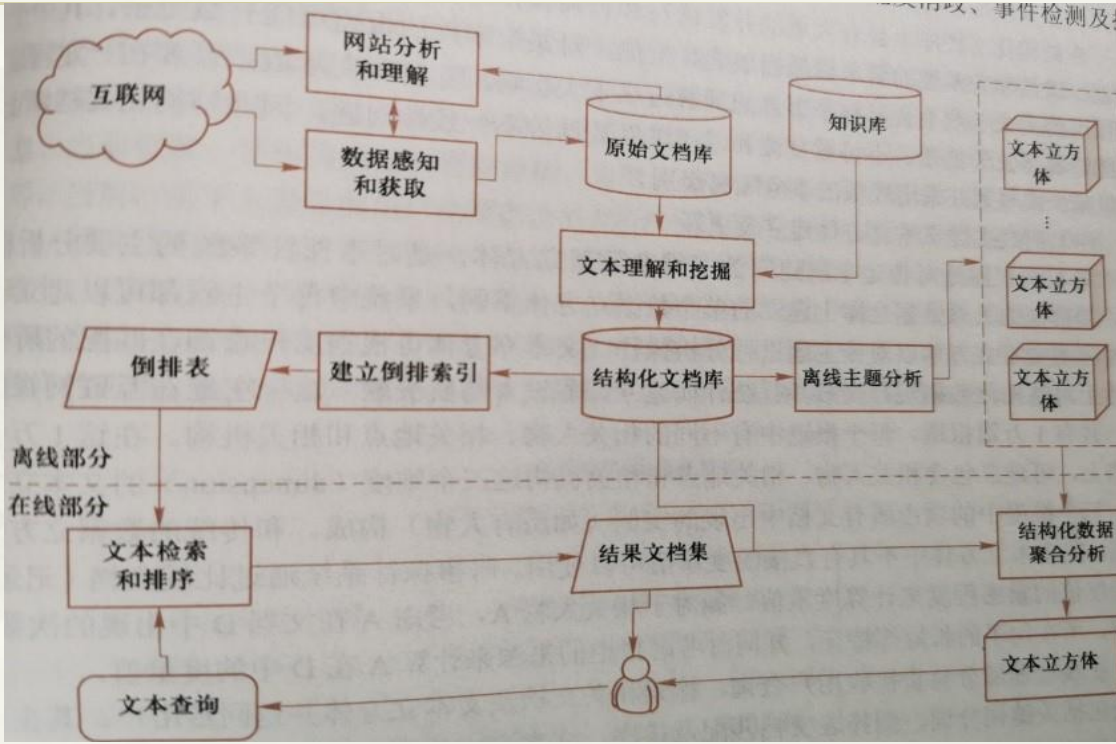
--互联网文本大数据管理与挖掘

* 基于大数据分析的用户建模

--基于大数据分析的用户建模

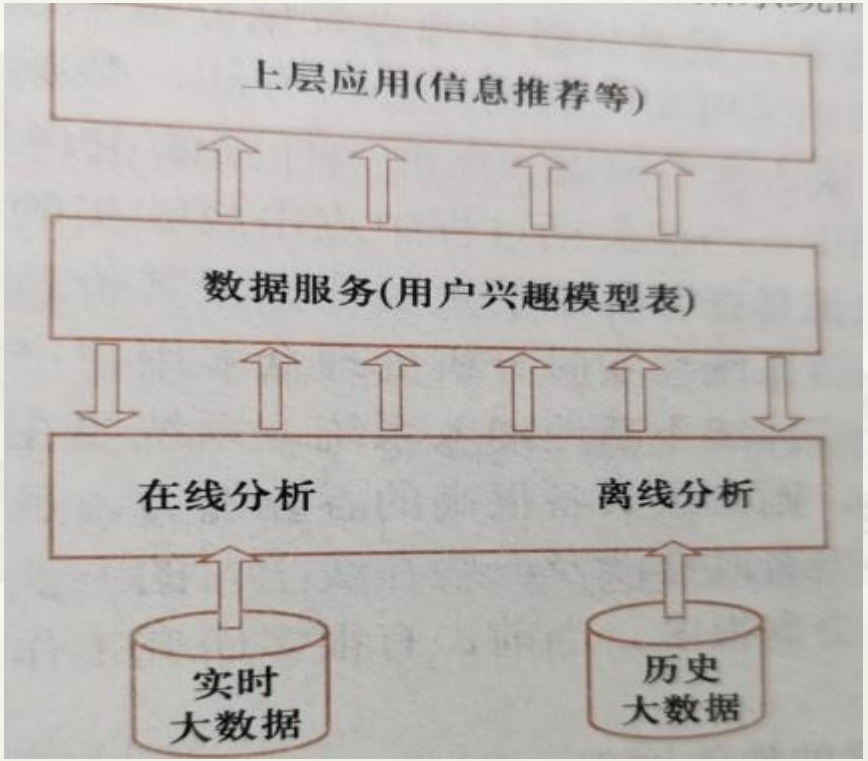
互联网文本大数据管理与挖掘

- 动态数据抓取
- 历史数据保留
- 数据深度智能分析
- 可视化展示
- 敏感信息实时获取
- 预定阈值报警



基于大数据分析的用户建模

- 根据用户创建的内容，浏览日志创建动态的用户描述文件。
- 根据用户的行为特征，兴趣爱好精准地进行个性化信息服务



大数据模型

Nosql 数据管理系统

* Non-relational
* Not only SQL

Newsql数据管理系统

* 介于关系模型与NOSQL模型之间

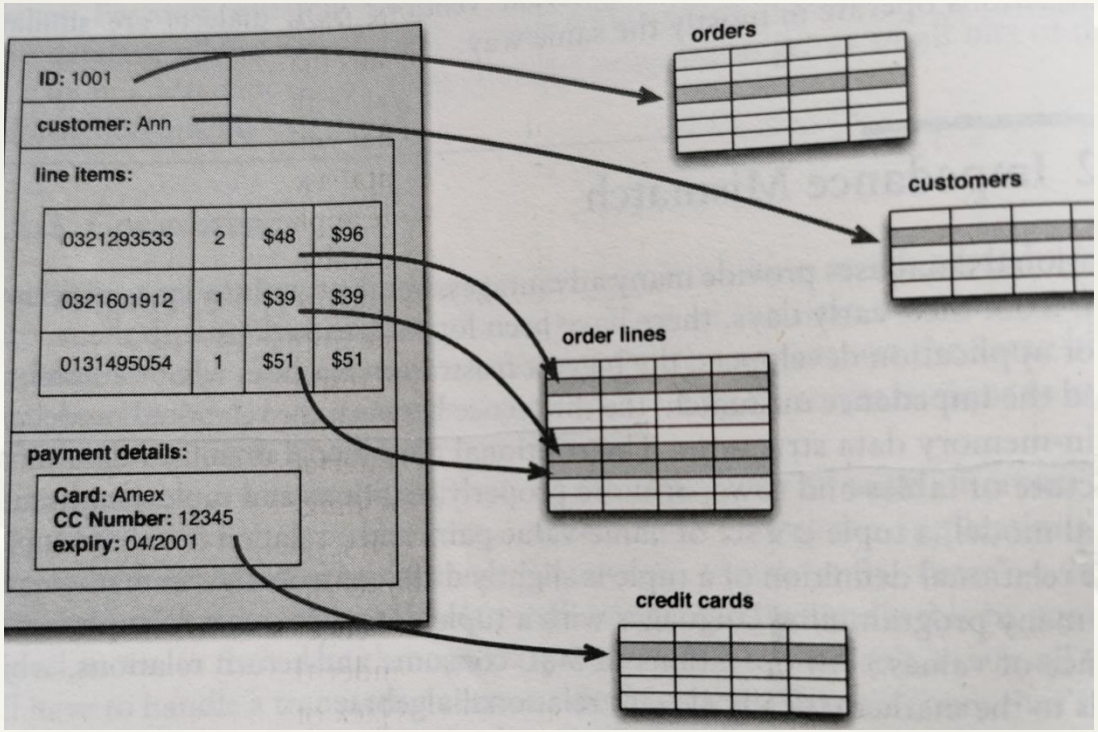
回顾: 关系模型

* 数据库存贮大量数据，数据具有一致性.
* 通过事务实现并发控制：提交或回滚
* 多个应用共享数据库
* 有一个标准模型（标准SQL语句等）

关系模型与实际应用

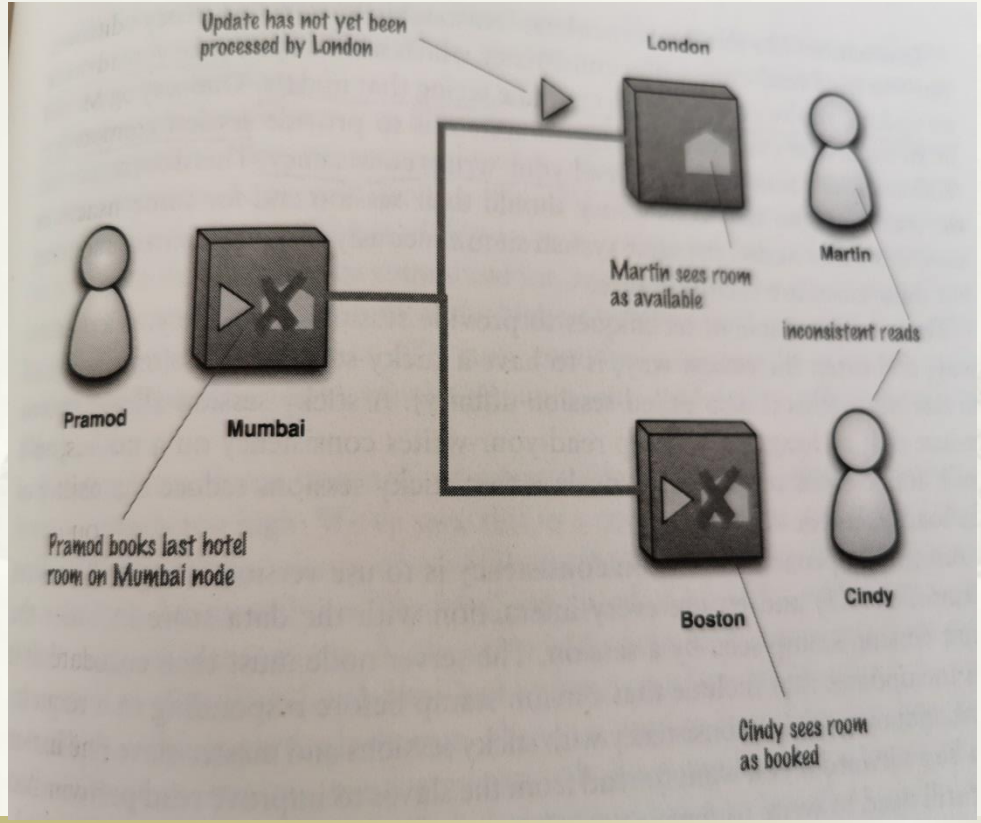
- 集成数据库阻碍了新应用的开发→应用数据库产生
- 大数据爆发需要大机器，多处理器，大的存贮空间→需要集群（a cluster of small machines）
- 分布式算法，可扩展性，混合存贮模式

互联网应用：聚合数据模型




```
{
 “order” :{
 “id” :1001,
 “Customer” :” Ann”,
 “items” :[{“itemID” :
” 0321293533”,,” num” :
2,” price” :$46,” total
” :$96}, …],
 “payment” :
{“card” :” Amex”,” CC
” :12345,” expiry” :
 “12/2020” }
```

互联网应用：酒店预定



* 假设现在酒店只剩下一间房，三个人同时在三个不同的地方预定。

NOSQL

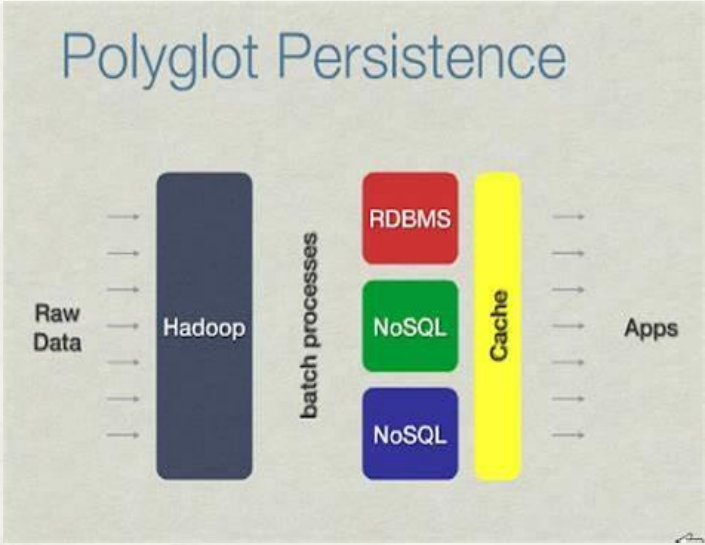
* 以牺牲事务机制和强一致性机制，来获取更好的分布式部署能力和横向扩展能力，创造出新的数据模型，使其在不同的应用场景下，对特定业务具有更强的粗粒性能。

NOSQL应用场景

1：数据模型比较简单；
2：需要灵活性更强的IT系统；
3：对数据库**性能要求较高**；
4：不需要高度的数据一致性；
5：对于给定key，比较容易映射复杂值的环境。

NOSQL 特点

* Not using the relational model
* Running well on clusters
* Open-source
* Built for the 21st century web estate
* Schemaless
* Polyglot persistence: using different data stores in different circumstances.



NOSQL 特点（续）

- 灵活的数据模型
- 可伸缩性强：分布式，横向扩展，**适合互联网应用的分布式特性**。
- **自动分片**：自动在多台服务器上分发数据，不需要应用程序增加额外的操作。
- **自动复制**：服务器自动对数据进行备份，复制存贮在多台服务器上，多个用户访问同一数据时，可以将用户请求分散到多台服务器中。
- **分布式处理**：数据逻辑分割（便于存放多个结点），物理分布（复制多个副本）

NOSQL 四种数据库

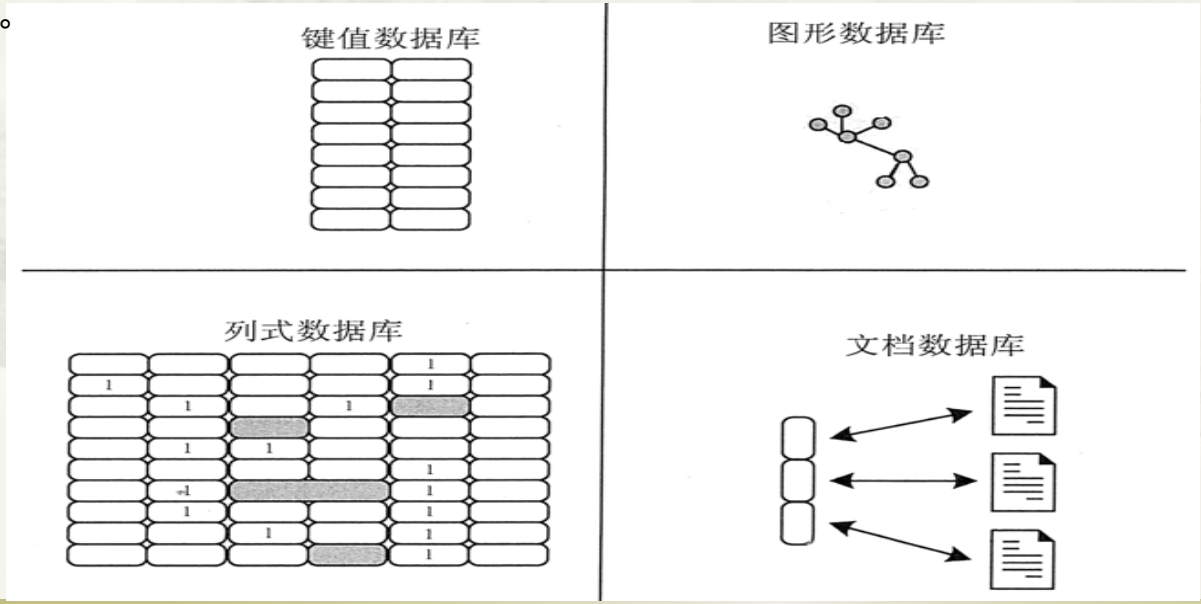
没有一个统一的架构，两种不同的NOSQL数据库差异程度较大

键值数据库：

列族数据库：每一行有关键字row key，每一行由多个列族组成，每个列族由多个列组成，列是（name:value）对

文档数据库：键值数据库的升级版，允许嵌套健值，可以根据文档内容创建索引

图形数据库：以节点、边、节点之间的关系来存贮复杂网络中的数据，在图形模式中，关系和节点本身就是数据。



NOSQL 数据模型: Key-Value 模型

- 键$k1$ 对应的值 $value=\{11,22,33\}$
- 键$k2$ 对应的值是一个字符串数组 {Name:Jim,Tel:1234}
- Key-Value 模型支持任意格式的值存储

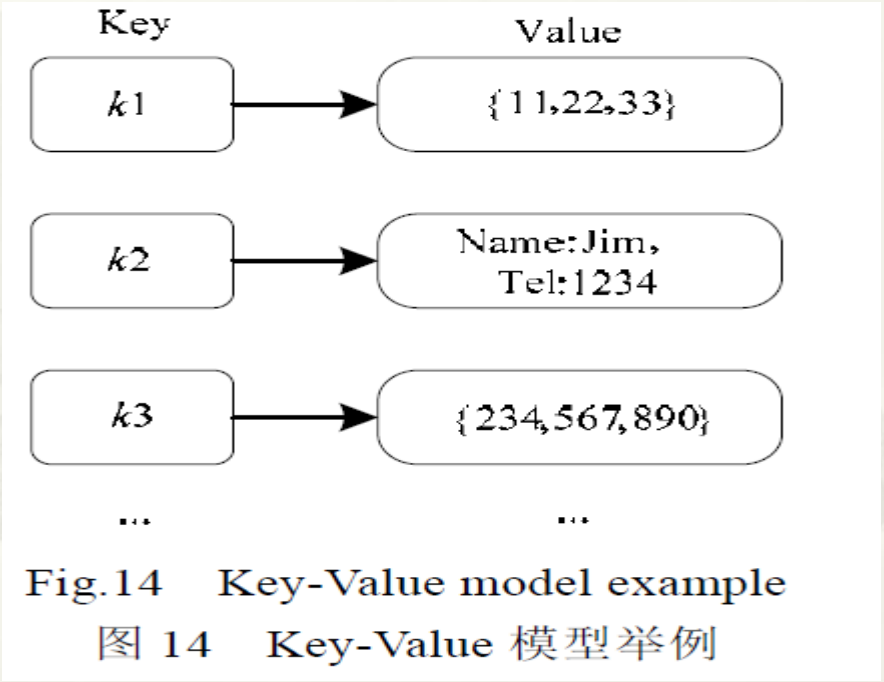


Fig.14 Key-Value model example

图 14 Key-Value 模型举例

NOSQL 数据模型: Key-Document 模型

- 数据用文档来表示。
- 面向集合：每个集合有一个唯一标识，存储在集合中的文档没有数量限制
- 无需定义模式。

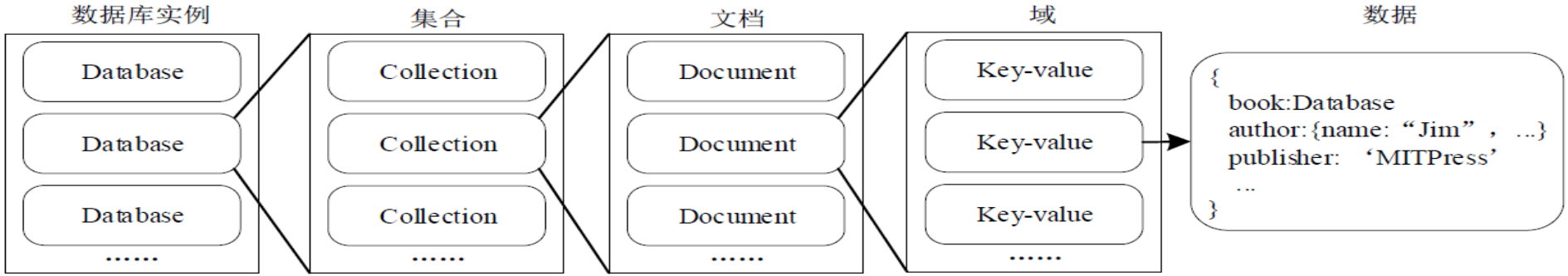


Fig.15 Key-Document model example

图 15 Key-Document 模型举例

Key-Column 模型（bigTable）

* 稀疏的、分布式的、持久化的多维排序图，并通过字典顺序来组织数据，支持动态扩展，以达到负载均衡.

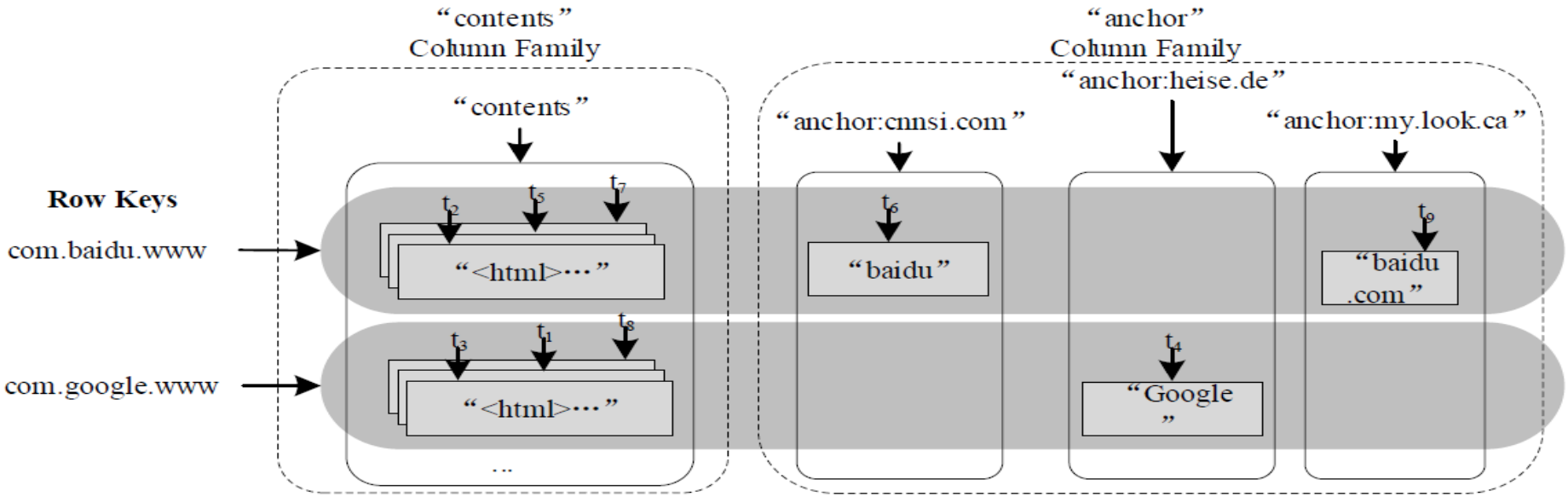


Fig.16 Key-Column model example

图 16 Key-Column 模型举例

三种NOSQL模型对比

Table 6 NoSQL database instance comparison

表 6 NoSQL 数据库实例对比

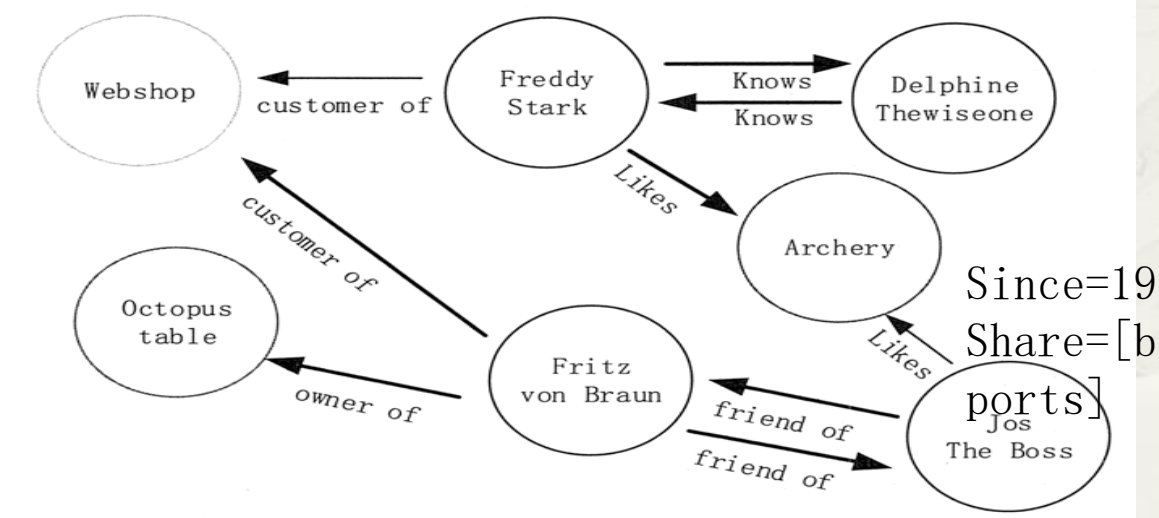
数据模型	Key-Value 模型			Key-Document 模型			Key-Column 模型	
数据库实例	Redis	Memcached	LevelDB	MongoDB	DynamoDB	CouchDB	HBase	Cassandra
实现语言	C	C	C++	C++	–	Erlang	Java	Java
二级索引	不支持	不支持	不支持	支持	支持	支持	不支持	受限制的
SQL	不支持	不支持	不支持	不支持	不支持	不支持	不支持	CQL
触发器	不支持	不支持	不支持	不支持	支持	支持	支持	支持
数据划分	分区	不支持	不支持	分区	分区	分区	分区	分区
数据副本	主从复制; 多主节点	不支持	不支持	主从复制	–	主\从复制; 主\主复制	可选择地 复制因子	可选择地 复制因子
外键	不支持	不支持	不支持	不支持	不支持	不支持	不支持	不支持
事务特性	支持	无	无	无	无	无	无	无
操作并发	支持	支持	支持	支持	支持	支持	支持	支持
数据持久	支持	不支持	支持	支持	支持	支持	支持	支持
内存计算	支持	支持	支持	支持	–	不支持	不支持	不支持

图数据库：Neo4J、Infinite Graph、OrientDB

- 储存实体和实体之间的关系.
- 边可以表示不同的特性
- 关系是有向的, 有意义的

适用场景:

- 在一些关系性强的数据中
- 推荐引擎



一致性模型

定义:

A consistency model determines the effect of concurrent operations on shared data as viewed by different clients of the system.

分布式系统中，存在很多不同的一致性模型，来保证数据项和副本的语义关系。

分布式系统的一致性模型:
Linearizability vs. sequential consistency

共同点:

- Should provide the behavior of a single copy 相同于单副本的操作结果
- A read operation returns the most recent write, regardless of the clients.
- All subsequent read ops should return the same result until the next write, regardless of the clients.

不同点:

* Linearizability cares about time. (in nontransactional systems)
* Sequential consistency cares about program order. (in transactional systems)

线性一致性模型: Linearizability

- “The most recent” & “all subsequent” - Determined by time.
- 任何一次读都能读到某个数据的最近一次写的数据。 -
- 系统中的所有进程,看到的操作顺序,都与全局时钟下的顺序一致。

顺序一致性: Sequential consistency

"The most recent" & "all subsequent"

* - Ops within the same client: Determined by time (program order)
* - Ops across clients: Not determined by time,

we can reorder them. - i.e., we just need to preserve the program order

NOSQL并发控制

- 问题：读脏数据, 不可重复读, 丢失修改等.
- 分布式系统(网络)发生问题：
  - 当两个人同时修改同一数据.
  - 一个用户读到了另一个用户不一致（修改操作的中间结果）的数据

强一致性：　更新后的数据能被不同进程访问到
弱一致性：　容忍后续部分或者全部访问不到
最终一致性：　经过一段时间能访问到更新后的数据.

如何实现呢?

问题：两个人同时修改同一数据.（更新一致性）

策略:

1. 悲观方法：避免同时发生
2. 乐观方法：允许同时修改，检查并采取行动.
3. 有条件修改：修改数据前先读数据，是否和以前一样，如果一样就修改。

问题：一个用户读到了另一个用户不一致（修改操作的中间结果）的数据

策略：

1. 最终一致性.
2. 读已之所写一致性（Read-your-writes consistency）：a client can write and then immediately read the new value. This can be difficult if the read and the write happen on different nodes.

客服端一致性的5个变种

从客户端来看，一致性主要指的是多并发访问时更新过的数据如何获取的问题。

- 因果一致性：进程A更新完数据通知进程B，则B能够获取进程A更新后的值。
- 读已之所写一致性：自己能访问到更新过的最新值。
- 会话一致性：系统能保证在同一个有效的会话中实现“读己之所写”的一致性。
- 单调读一致性：如果一个进程从系统中读取出一个数据项的某个值后，那么系统对于该进程后续的任何数据访问都不应该返回更旧的值。
- 单调写一致性：同上。

最终一致性：在实际系统中，将上述策略混合以构建一个具有最终一致性的分布式系统

服务器端一致性

指更新如何复制分布到整个系统，以保证数据最终一致

- N: 数据复制份数
- W:更新数据时需要写操作的节点数
- R:读取数据时需要读取的节点数

- 强一致性：W+R>N
- 弱一致性：W+R<=N
- 如果N=W　R=1：任何一个写节点失效,都会导致写失败，可用性降低，但是数据分布的N个节点同步写入，可以保证强一致性。
- N=R W=1：则需要一个节点写入成功，写性能和可用性都比较高，但读取其他节点的进程可能不能获取更新后的数据，弱一致性。

NOSQL 数据库事务管理：CAP

- 一致性C：all nodes see the same data at the same time.更新操作成功后，所有节点在同一时间的数据完全一致
- 可用性A：reads and writes always succeed.用户访问数据时，系统是否能在响应时间返回结果。
- 分区容错性P: the system continues to operate despite arbitrary message loss or failure of part of the system. 遇到某节点或网络分区故障时，仍然能够对外提供满足一致性和可用性服务。

CAP原则：分布式系统只能满足上述2个特性：CA,CP,AP。

CAP 原则

对NOSQL，分区容错性不能牺牲，一般有：

* CP模式：分区容忍性,同时对数据一致性要求较高。例如bigtable，Hbase,Redis等
* AP模式：主要以实现最终一致性来确保可用性和分区容忍性,弱化对数据的一致要求

NOSQL数据库事务管理： BASE

BASE 理论是针对NOSQL数据库而言，对CAP理论中的一致性和可用性进行权衡的结果：

Basically Available：保证核心功能可用，损失部分可用性
Soft-state：允许不同节点副本之间存在暂时不一致情况。
Eventually Consistent：不需要实时保证数据副本一致，最终一致即可。

总结：**通过弱化一致性，提高系统的伸缩性、可靠性和可用性。**

讨论

你能否举出实际应用环境中，应用CAP 或BASE 原则的例子吗？

Categorization of the Representative NoSQL Stores Based on the CAP Theorem

CAP principle	NoSQL store	The sacrifice of CAP consistency	The sacrifice of CAP availability
CP	Scalaris	No sacrifice	It is sacrificed during the DHT-based Paxos commit, when the majority of *participants* or *transaction managers* are not available.
AP	Amazon Dynamo	It supports a tunable eventual consistency.	No sacrifice
	Apache Cassandra	It supports a tunable eventual consistency.	No sacrifice
	COPS	It supports only causal$^+$ consistency across clusters.	No sacrifice
	CouchDB	There is no support for multi-document transactions	No sacrifice
P	Yahoo PNUTS	Lack of bundled write operations that span multiple data items.	Unavailability of the primary copy during the primary election.
	Google Bigtable	Lack of cross-row, multikey transactions.	Bigtable relies on the replication strategy of GFS, which is restricted within a datacenter. In addition, since Chubby is a *CP* system, in the case of network partitioning, clients in the minority side of the partition may not have access to data.
	IBM Spinnaker	Lack of cross-row, multikey transactions.	Due to the use of Paxos-based replication, in the case of network partitioning, writes and *strongly consistent* reads are unavailable for clients in the minority side of the partition.
	HyperDex	Lack of cross-row, multikey transactions.	When the number of replica node failures (in the block of replicas corresponding to each *subspace*) is more than a threshold.
	Scatter	Lack of cross-row, multikey transactions.	Owing to the use of Paxos-based replication. However, as long as a majority of nodes of a *group* remains alive, the used 2PC protocol is not blocking.
	Walter	Causal consistency is enforced on transactions across datacenters.	Failure of a datacenter *D* causes unavailability of writes on those objects whose *preferred datacenter* is *D*.
	MongoDB	Lack of cross-document, multikey transactions.	In the case of network partitions, nodes belonging to the minority side of a partitioned *replica set* are not available for *write* operations.
	Couchbase	There is no support for multi-document transactions	Similar to MongoDB.
	Microsoft Trinity	Lack of multikey transactions.	It is vulnerable to loss of data in the case of the failure of the datacenter where the Trinity cluster is deployed.
	Neo4j	Eventual consistency is enforced on operations across datacenters.	In the case of network partitions, nodes belonging to the minority side of the network are not available for *write* operations.

NOSQL 总结

- trade off consistency versus latency: to get good consistency, you need to involve many nodes in data operations, but this increases latency.
- Trade off durability versus latency: survive failures with replicated data.
- Trade off availability versus consistency.

NEWSQL模型

* 使用SQL语言作为应用之间交互的主要机制。
* 遵循ACID
* 使用“无锁”的并发机制。
* 结合NOSQL和传统SQL系统的优点

NEWSQL 种类

1. 新型架构：使用一个全新的系统来实现
2. 透明的数据分片中间件：使用自动分片的中间件将数据库分成多个部分，并存贮到多个单节点机器组成的集群中.
3. Database-as-a-service：使用云服务提供商提供的系统，它们负责维护所有的数据库物理机及其配置，包括系统优化（例如缓冲池调整），复制，以及备份。

1. New architectures

新结构的NEWSQL系统具有以下特点：

- Distributed architectures
- Shared-nothing resources
- Multi-node concurrent control
- fault tolerance through replication
- Distributed query processing
- allows the DBMS to “send the query to the data”

2. TransparentShardingMiddleware

* 把一个数据库中数据分成几个部分，分别存放在不同的结点，每个结点可以：

(1) runs the same DBMS

(2) only has a portion of the overall database,

(3) is not meant to be accessed and updated independently by separate applications.

这种分片对用户和开发人员是透明的，即自动分片。

3. Database-as-a-Service

- 云计算的数据库服务商负责数据库的维护和运行。
- 基于新型架构的DBaaS是NewSQL
- Amazon’s Aurora 是其中的一个例子

NEWSQL 特点

* Main memory system: 可以把所有数据库装入内存，有技术可以把数据库的子集转移到外存。
* Partitioning/sharding: 数据库数据水平和垂直分区，不同节点上分布式处理
* Concurrency control: 保持事务的原子性和隔离性。使用timestamp ordering (TO) concurrency control, 和 MULTI-VERSION concurrency control (MVCC)，二段锁和多版本控制结合起来的方案

NEWSQL 特点 (续)

* 副本设计策略：强一致性和弱一致性：active-active replication（每个副本节点同时处理更新） and active-passive replication. （首先在一个节点更新，然后，DBMS把处理状态转发到其它复制节点）
* 出错恢复：恢复时有两种方法：一是后备结点从自身存贮设备输入它最后一个checkpoint and WAL,然后再把其它结点的日志文件装入二是后备结点重新装入一个用于恢复的新日志.

NEWSQL 的优点

- 数据库分区减少了系统的通信开销，从而可以轻松地访问数据。
- 即使出现系统故障或错误，ACID事务也可以确保数据的完整性。
- NewSQL数据库可以处理复杂的数据。
- NewSQL系统具有高度可伸缩性。

NEWSQL system overview

		Year Released	Main Memory Storage	Partitioning	Concurrency Control	Replication	Summary
NEW ARCHITECTURES	**Clustrix** [6]	2006	No	Yes	MVCC+2PL	Strong+Passive	MySQL-compatible DBMS that supports shared-nothing, distributed execution.
	CockroachDB [7]	2014	No	Yes	MVCC	Strong+Passive	Built on top of distributed key/value store. Uses software hybrid clocks for WAN replication.
	Google Spanner [24]	2012	No	Yes	MVCC+2PL	Strong+Passive	WAN-replicated, shared-nothing DBMS that uses special hardware for timestamp generation.
	H-Store [8]	2007	Yes	Yes	TO	Strong+Active	Single-threaded execution engines per partition. Optimized for stored procedures.
	HyPer [9]	2010	Yes	Yes	MVCC	Strong+Passive	HTAP DBMS that uses query compilation and memory efficient indexes.
	MemSQL [11]	2012	Yes	Yes	MVCC	Strong+Passive	Distributed, shared-nothing DBMS using compiled queries. Supports MySQL wire protocol.
	NuoDB [14]	2013	Yes	Yes	MVCC	Strong+Passive	Split architecture with multiple in-memory executor nodes and a single shared storage node.
	SAP HANA [55]	2010	Yes	Yes	MVCC	Strong+Passive	Hybrid storage (rows + cols). Amalgamation of previous TREX, P*TIME, and MaxDB systems.
	VoltDB [17]	2008	Yes	Yes	TO	Strong+Active	Single-threaded execution engines per partition. Supports streaming operators.

NEWSQL DBMS overview

NEW ARCHITECTURES	**CockroachDB** [7]	2014	No	Yes	MVCC	Strong+Passive	Built on top of distributed key/value store. Uses software hybrid clocks for WAN replication.
	Google Spanner [24]	2012	No	Yes	MVCC+2PL	Strong+Passive	WAN-replicated, shared-nothing DBMS that uses special hardware for timestamp generation.
	H-Store [8]	2007	Yes	Yes	TO	Strong+Active	Single-threaded execution engines per partition. Optimized for stored procedures.
	HyPer [9]	2010	Yes	Yes	MVCC	Strong+Passive	HTAP DBMS that uses query compilation and memory efficient indexes.
	MemSQL [11]	2012	Yes	Yes	MVCC	Strong+Passive	Distributed, shared-nothing DBMS using compiled queries. Supports MySQL wire protocol.
	NuoDB [14]	2013	Yes	Yes	MVCC	Strong+Passive	Split architecture with multiple in-memory executor nodes and a single shared storage node.
	SAP HANA [55]	2010	Yes	Yes	MVCC	Strong+Passive	Hybrid storage (rows + cols). Amalgamation of previous TREX, P*TIME, and MaxDB systems.
	VoltDB [17]	2008	Yes	Yes	TO	Strong+Active	Single-threaded execution engines per partition. Supports streaming operators.
MIDDLEWARE	**AgilData** [1]	2007	No	Yes	MVCC+2PL	Strong+Passive	Shared-nothing database sharding over single-node MySQL instances.
	MariaDB MaxScale [10]	2015	No	Yes	MVCC+2PL	Strong+Passive	Query router that supports custom SQL rewriting. Relies on MySQL Cluster for coordination.
	ScaleArc [15]	2009	No	Yes	Mixed	Strong+Passive	Rule-based query router for MySQL, SQL Server, and Oracle.
DBAAS	**Amazon Aurora** [3]	2014	No	No	MVCC	Strong+Passive	Custom log-structured MySQL engine for RDS.
	ClearDB [5]	2010	No	No	MVCC+2PL	Strong+Active	Centralized router that mirrors a single-node MySQL instance in multiple data centers.

三种模型的对比

Table 1 Comparison of RDBMS, NoSQL, and NewSQL features

表 1 RDBMS、NoSQL 和 NewSQL 特点比较

	RDBMS	NoSQL	NewSQL
SQL	支持	不支持	支持
宿主机	单机	多机/分布式	多机/分布式
类型	关系型	非关系型	关系型
模式	表	key-(value,column,document)	二者都支持
物理存储	磁盘+缓存	磁盘+缓存	磁盘+缓存
特性	ACID	CAP,BASE	ACID
查询复杂度	低	高	高
一致性	高	最终一致性	高
可用性	故障转移	高	高
可扩展性	垂直扩展	水平扩展	水平扩展
复制	可配置	可配置	自动
安全性	高	低	低
大数据处理	支持,但效率低	支持	充分支持
OLTP	不完全支持	支持	充分支持

大数据管理的新格局

- 面向**操作性**应用：基于行存贮的关系数据库系统，并行数据库系统，实时计算的内存数据库系统，NoSQL 系统，以及结合nosql 和关系的新型系统（VoltDB）
- 面向**分析型**应用：列存贮数据库（MonetDB）和基于列存贮技术的内存数据库（MonetDB,VectorWise,Hana），以及采用MapReduced 技术，面向分析应用的Nosql系统。

数据管理技术新格局

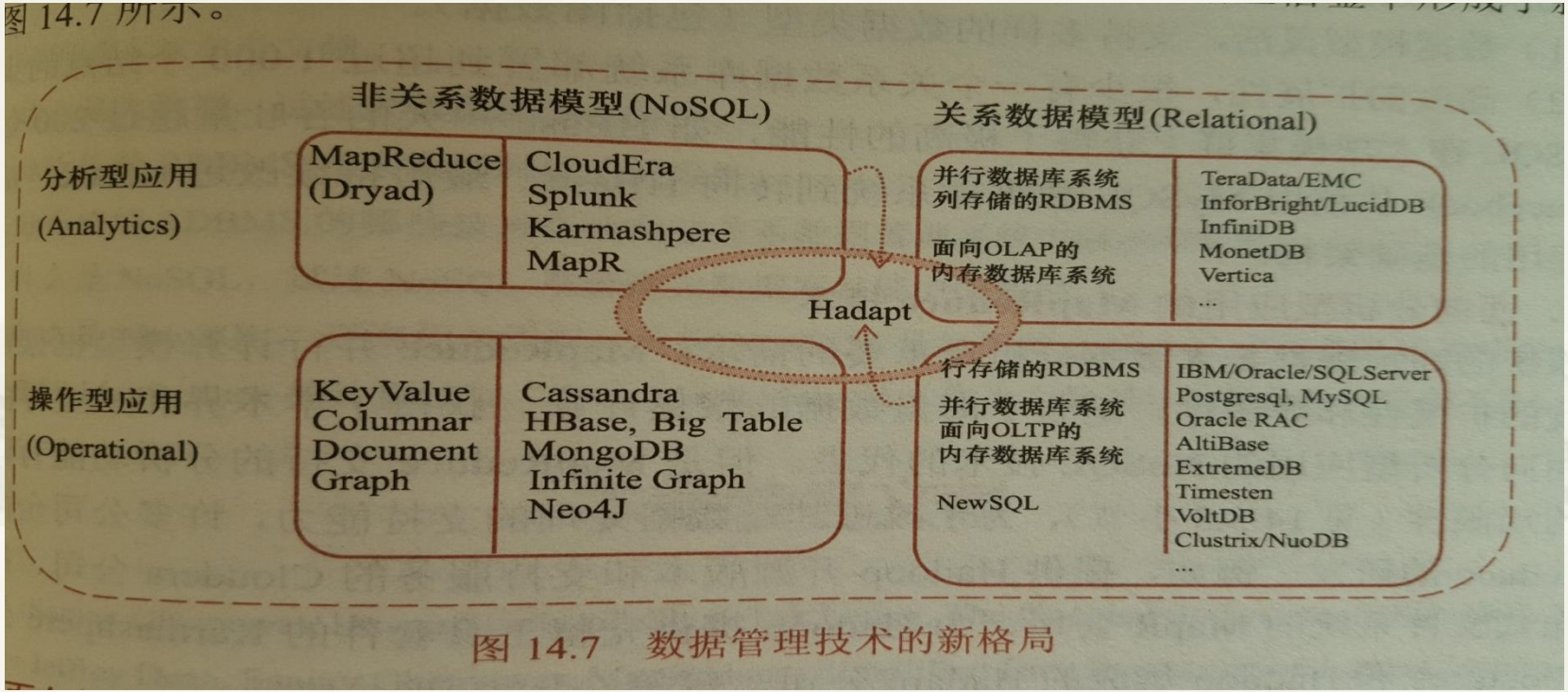


图 14.7　数据管理技术的新格局

大数据管理的新格局

* 面向操作型应用的数据库技术

基于行存贮的关系数据库系统，面向实时计算的内存数据库系统，形成新的NEWSQL系统

* 面向分析型应用的关系数据库技术

面向分析的列存贮数据库和内存数据库，内存数据库利用大内存，多核CPU等硬件系统。

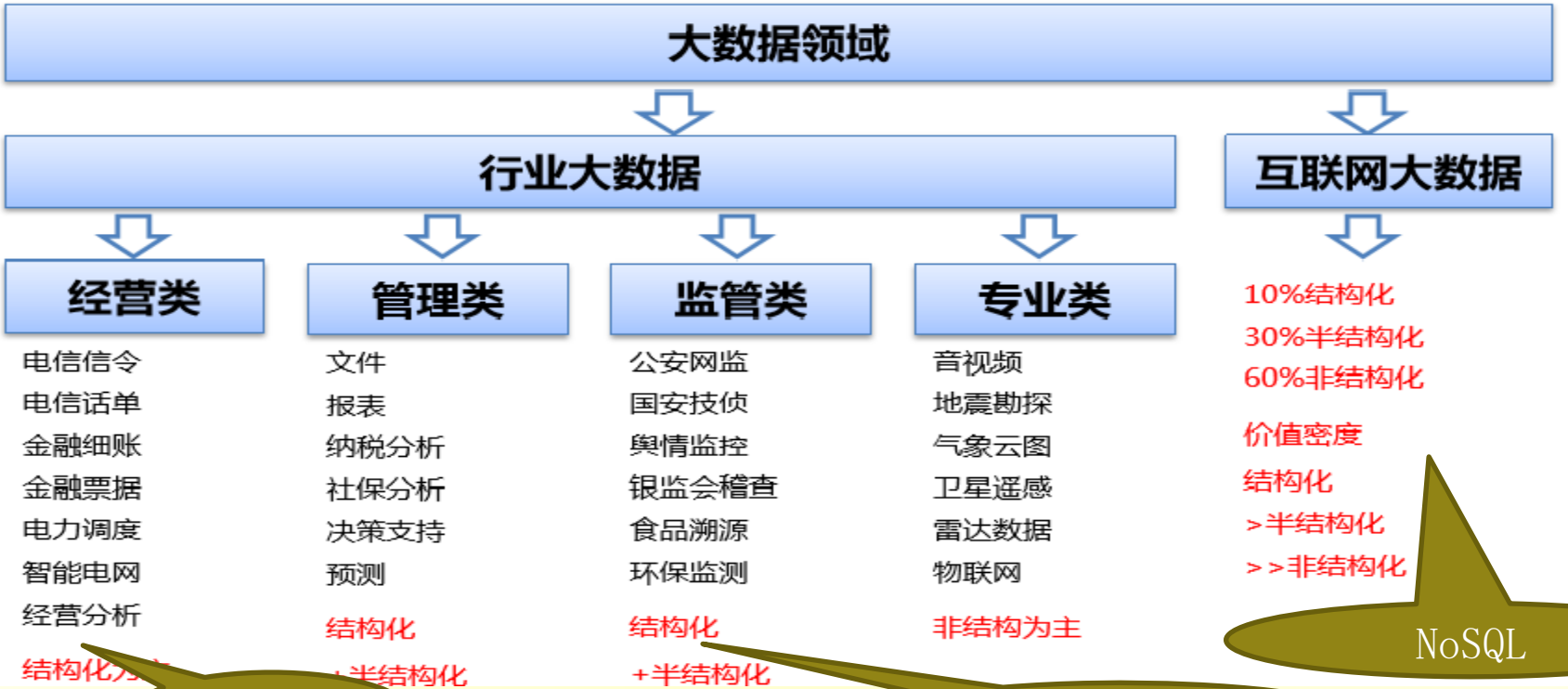
* 面向操作型应用的NOSQL技术

NOSQL系统，数据模型灵活，扩展性好

* 面向分析型的MAPreduce技术

并行计算的框架，简单，高度的扩展性和容错性，适合海量数据的聚集计算。

大数据的宏观视图：行业与互联网大数据



新型数据管理系统分类

系统类型	代表性系统	主要解决的问题
分布式数据管理系统	MongoDB,Redis,Cassandra,Spanner,Oceanbase	数据的规模大(volume)
流数据管理系统	STREAM,Aurora,TelegraphCQ,NiagaraCQ,Gigascope	数据的变化快(velocity)
图数据管理系统	Pregel,Giraph,PowerGraph,GraphChi,Xstream,Giraph+	数据的种类杂(variety)
时空数据管理系统	SpatialHadoop,Simb,OceanRT,DITA,SECONDO	数据的种类杂(variety)
众包数据管理系统	CrowdDB,CDB,Deco,Qurk,DOCS,gMission	数据的价值密度低(value)

参考文献

* 大数据管理：概念、技术与挑战
* 数据模型及其发展历程
* 新型数据管理系统研究进展与趋势