第6章 NoSQL数据库与云数据库

高琰

目录

概念演变 →

Not only SQL

最初表示“反SQL”运动
用新型的非关系数据库取代关系数据库

现在表示关系和非关系型数据库各有优缺点
彼此都无法互相取代

通常，NoSQL数据库具有以下几个特点：

（1）灵活的可扩展性

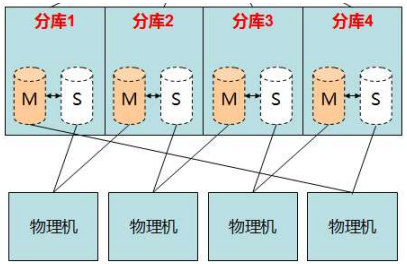
（2）灵活的数据模型

（3）与云计算紧密融合

6.1 NoSQL概述-兴起的原因

海量数据？MySQL集群是否可以完全解决问题？

- **复杂性**：部署、管理、配置很复杂
- **数据库复制**：MySQL主备之间采用复制方式，只能是异步复制，当主库压力较大时可能产生较大延迟，主备切换可能会丢失最后一部分更新事务，这时往往需要人工介入，备份和恢复不方便
- **扩容问题**：如果系统压力过大需要增加新的机器，这个过程涉及数据重新划分，整个过程比较复杂，且容易出错
- **动态数据迁移问题**：如果某个数据库组压力过大，需要将其中部分数据迁移出去，迁移过程需要总控节点整体协调，以及数据库节点的配合。这个过程很难做到自动化



NoSQL与关系数据库的比较

表5-1 NoSQL和关系数据库的简单比较

比较标准	RDBMS	NoSQL	备注
数据库原理	完全支持	部分支持	RDBMS有关系代数理论作为基础 NoSQL没有统一的理论基础
数据规模	大	超大	RDBMS很难实现横向扩展，纵向扩展的空间也比较有限，性能会随着数据规模的增大而降低 NoSQL可以很容易通过添加更多设备来支持更大规模的数据
数据库模式	固定	灵活	RDBMS需要定义数据库模式，严格遵守数据定义和相关约束条件 NoSQL不存在数据库模式，可以自由灵活定义并存储各种不同类型的数据
查询效率	快	可以实现高效的简单查询，但是不具备高度结构化查询等特性，复杂查询的性能不尽人意	RDBMS借助于索引机制可以实现快速查询（包括记录查询和范围查询） 很多NoSQL数据库没有面向复杂查询的索引，虽然NoSQL可以使用MapReduce来加速查询，但是，在复杂查询方面的性能仍然不如RDBMS

6.1 NoSQL概述-兴起的原因

2、“One size fits all”模式很难适用于截然不同的业务场景

- 关系模型作为统一的数据模型既被用于数据分析，也被用于在线业务。但这两者一个强调高吞吐，一个强调低延时，已经演化出完全不同的架构。用同一套模型来抽象显然是不合适的
- Hadoop就是针对数据分析
- MongoDB、Redis等是针对在线业务，两者都抛弃了关系模型

NoSQL与关系数据库的比较

表5-1 NoSQL和关系数据库的简单比较（续）

比较标准	RDBMS	NoSQL	备注
一致性	强一致性	弱一致性	RDBMS严格遵守事务ACID模型，可以保证事务强一致性 很多NoSQL数据库放松了对事务ACID四性的要求，而是遵守BASE模型，只能保证最终一致性
数据完整性	容易实现	很难实现	任何一个RDBMS都可以很容易实现数据完整性，比如通过主键或者非空约束来实现实体完整性，通过主键、外键来实现参照完整性，通过约束或者触发器来实现用户自定义完整性 但是，在NoSQL数据库却无法实现
扩展性	一般	好	RDBMS很难实现横向扩展，纵向扩展的空间也比较有限 NoSQL在设计之初就充分考虑了横向扩展的需求，可以很容易通过添加廉价设备实现扩展
可用性	好	很好	RDBMS在任何时候都以保证数据一致性为优先目标，其次才是优化系统性能，随着数据规模的增大，RDBMS为了保证严格的一致性，只能提供相对较弱的可用性 大多数NoSQL都能提供较高的可用性

NoSQL与关系数据库的比较

表5-1 NoSQL和关系数据库的简单比较（续）

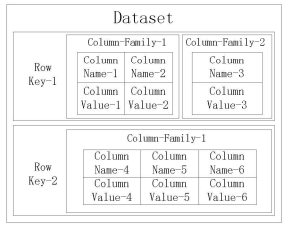
比较标准	RDBMS	NoSQL	备注
标准化	是	否	RDBMS已经标准化（SQL） NoSQL还没有行业标准，不同的NoSQL数据库都有自己的查询语言，很难规范应用程序接口 StoneBraker认为：NoSQL缺乏统一查询语言，将会拖慢NoSQL发展
技术支持	高	低	RDBMS经过几十年的发展，已经非常成熟，Oracle等大型厂商都可以提供很好的技术支持 NoSQL在技术支持方面仍然处于起步阶段，还不成熟，缺乏有力的技术支持
可维护性	复杂	复杂	RDBMS需要专门的数据库管理员(DBA)维护 NoSQL数据库虽然没有DBMS复杂，也难以维护

6.2 NoSQL的四大类型

NoSQL数据库虽然数量众多，但是，归结起来，典型的NoSQL数据库通常包括键值数据库、列族数据库、文档数据库和图形数据库

Key_1	Value_1
Key_2	Value_2
Key_3	Value_1
Key_4	Value_3
Key_5	Value_2
Key_6	Value_1
Key_7	Value_4
Key_8	Value_3

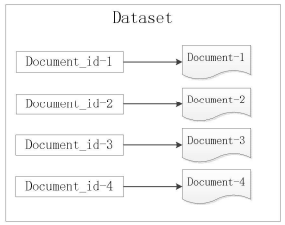
键值数据库



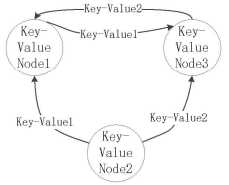
列族数据库

02 NoSQL的类型

6.4 NoSQL的四大类型



文档数据库

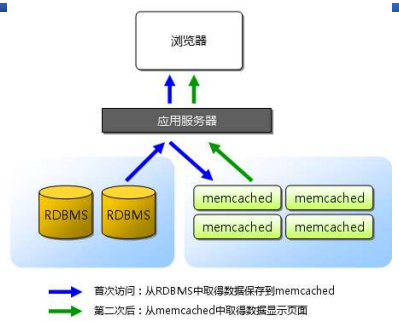


图形数据库

6.4 NoSQL的四大类型

文档数据库	图数据库
Couchbase MarkLogic mongoDB	Neo4j InfiniteGraph The Distributed Graph Database
键值数据库	**列族数据库**
redis amazon DynamoDB AEROSPIKE riak	accumulo HYPERTABLE Cassandra APACHE HBASE Amazon SimpleDB

5.4.1 键值数据库



键值数据库成为理想的缓冲层解决方案

Redis有时候会被人们称为“强化版的Memcached”
支持持久化、数据恢复、更多数据类型

5.4.1 键值数据库

相关产品	Redis、Riak、SimpleDB、Chordless、Scalaris、Memcached
数据模型	键/值对 键是一个字符串对象 值可以是任意类型的数据，比如整型、字符型、数组、列表、集合等
典型应用	涉及频繁读写、拥有简单数据模型的应用 内容缓存，比如会话、配置文件、参数、购物车等 存储配置和用户数据信息的移动应用
优点	扩展性好，灵活性好，大量写操作时性能高
缺点	无法存储结构化信息，条件查询效率较低
不适用情形	不是通过键而是通过值来查：键值数据库根本没有通过值查询的途径 需要存储数据之间的关系：在键值数据库中，不能通过两个或两个以上的键来关联数据 需要事务的支持：在一些键值数据库中，产生故障时，不可以回滚
使用者	百度云数据库（Redis）、GitHub（Riak）、BestBuy（Riak）、Twitter（Redis和Memcached）、StackOverFlow（Redis）、Instagram （Redis）、Youtube（Memcached）、Wikipedia（Memcached）

5.4.2 列族数据库

相关产品	BigTable、HBase、Cassandra、HadoopDB、GreenPlum、PNUTS
数据模型	列族
典型应用	分布式数据存储与管理 数据在地理上分布于多个数据中心的应用程序 可以容忍副本中存在短期不一致情况的应用程序 拥有动态字段的应用程序 拥有潜在大量数据的应用程序，大到几百TB的数据
优点	查找速度快，可扩展性强，容易进行分布式扩展，复杂性低
缺点	功能较少，大都不支持强事务一致性
不适用情形	需要ACID事务支持的情形，Cassandra等产品就不适用
使用者	Ebay（Cassandra）、Instagram（Cassandra）、NASA（Cassandra）、Twitter（Cassandra and HBase）、Facebook（HBase）、Yahoo!（HBase）

5.4.3 文档数据库

“文档”其实是一个数据记录，这个记录能够对包含的数据类型和内容进行“自我描述”。XML文档、HTML文档和JSON 文档就属于这一类。SequoiaDB就是使用JSON格式的文档数据库，它的存储的数据是这样的：

```
{
“ID”:1,
“NAME”: “SequoiaDB”,
“Tel”: {
      “Office”: “123123”, “Mobile”: “132132132”
    }
“Addr”: “China,GZ”
}
```

关系数据库：

必须有schema信息才能理解数据的含义

学生（学号，姓名，性别，年龄，系，年级）

（1001，张三，男，20，计算机，2002）

一个XML文档：

```
<configuration>
<property>
<name>hbase.rootdir</name>
<value>hdfs://localhost:9000/hbase</value>
</property>
</configuration>
```

5.4.3 文档数据库

相关产品	MongoDB、CouchDB、Terrastore、ThruDB、RavenDB、SisoDB、RaptorDB、CloudKit、Perservere、Jackrabbit
数据模型	键/值 值（value）是版本化的文档
典型应用	存储、索引并管理面向文档的数据或者类似的半结构化数据 比如，用于后台具有大量读写操作的网站、使用JSON数据结构的应用、使用嵌套结构等非规范化数据的应用程序
优点	性能好（高并发），灵活性高，复杂性低，数据结构灵活 提供嵌入式文档功能，将经常查询的数据存储在同一个文档中 既可以根据键来构建索引，也可以根据内容构建索引
缺点	缺乏统一的查询语法
不适用情形	在不同的文档上添加事务。文档数据库并不支持文档间的事务，如果对这方面有需求则不应该选用这个解决方案
使用者	百度云数据库（MongoDB）、SAP （MongoDB）、Codecademy （MongoDB）、Foursquare （MongoDB）、NBC News （RavenDB）

5.4.3 文档数据库

```
{
“ID”:1,
“NAME”: “SequoiaDB”,
“Tel”: {
      “Office”: “123123”, “Mobile”: “132132132”
    }
“Addr”: “China,GZ”
}
```

- 数据是不规则的，每一条记录包含了所有的有关“SequoiaDB”的信息而没有任何外部的引用，这条记录就是“自包含”的
- 这使得记录很容易完全移动到其他服务器，因为这条记录的所有信息都包含在里面了，不需要考虑还有信息在别的表没有一起迁移走
- 同时，因为在移动过程中，只有被移动的那一条记录（文档）需要操作，而不像关系型中每个有关联的表都需要锁住来保证一致性，这样一来ACID的保证就会变得更快速，读写的速度也会有很大的提升

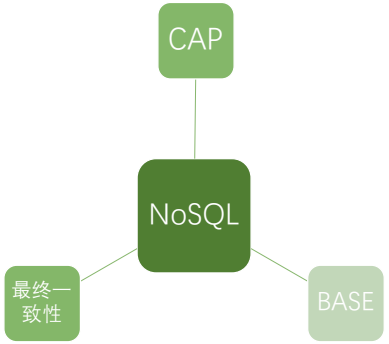
5.4.4 图形数据库

相关产品	Neo4J、OrientDB、InfoGrid、Infinite Graph、GraphDB
数据模型	图结构
典型应用	专门用于处理具有高度相互关联关系的数据，比较适合于社交网络、模式识别、依赖分析、推荐系统以及路径寻找等问题
优点	灵活性高，支持复杂的图形算法，可用于构建复杂的关系图谱
缺点	复杂性高，只能支持一定的数据规模
使用者	Adobe（Neo4J）、Cisco（Neo4J）、T-Mobile（Neo4J）

5.4.5 不同类型数据库比较分析

- **MySQL**产生年代较早，而且随着LAMP大潮得以成熟。尽管其没有什么大的改进，但是新兴的互联网使用的最多的数据库
- **MongoDB**是个新生事物，提供更灵活的数据模型、异步提交、地理位置索引等五花十色的功能
- **HBase**是个“仗势欺人”的大象兵。依仗着Hadoop的生态环境，可以有很好的扩展性。但是就像象兵一样，使用者需要养一头大象(Hadoop),才能驱使他
- **Redis**是键值存储的代表，功能最简单。提供随机数据存储。就像一根棒子一样，没有多余的构造。但是也正是因此，它的伸缩性特别好。就像悟空手里的金箍棒，大可捅破天，小能成缩成针

5.5 NoSQL的三大基石



03

NoSQL三大基石

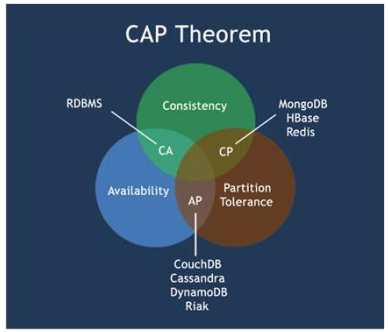
5.5.1 CAP

所谓的CAP指的是：

- C（Consistency）：一致性，是指任何一个读操作总是能够读到之前完成的写操作的结果，也就是在分布式环境中，多点的数据是一致的，或者说，所有节点在同一时间具有相同的数据
- A:（Availability）：可用性，是指快速获取数据，可以在确定的时间内返回操作结果，保证每个请求不管成功或者失败都有响应；
- P（Tolerance of Network Partition）：分区容忍性，是指当出现网络分区的情况时（即系统中的一部分节点无法和其他节点进行通信），分离的系统也能够正常运行，也就是说，系统中任意信息的丢失或失败不会影响系统的继续运作。

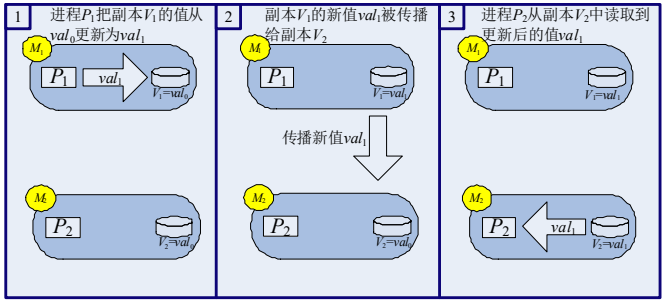
5.5.1 CAP

CAP理论告诉我们，一个分布式系统不可能同时满足一致性、可用性和分区容忍性这三个需求，最多只能同时满足其中两个，正所谓“鱼和熊掌不可兼得”。



5.5.1 CAP

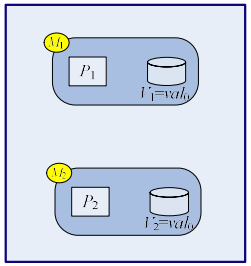
一个牺牲一致性来换取可用性的实例



（b）正常执行过程

5.5.1 CAP

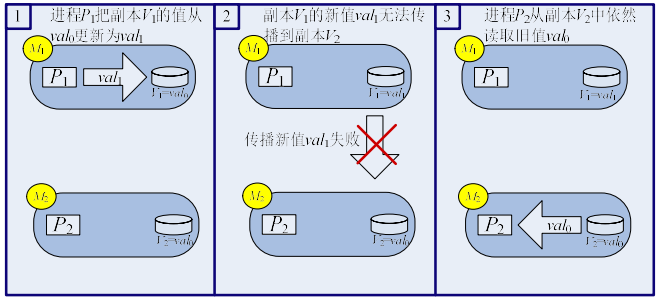
一个牺牲一致性来换取可用性的实例



（a）初始状态

5.5.1 CAP

一个牺牲一致性来换取可用性的实例



(c) 更新传播失败时的执行过程

5.5.1 CAP

当处理CAP的问题时，可以有几个明显的选择：

1.CA：也就是强调一致性（C）和可用性（A），放弃分区容忍性（P），最简单的做法是把所有与事务相关的内容都放到同一台机器上。很显然，这种做法会严重影响系统的可扩展性。传统的关系数据库（MySQL、SQL Server和PostgreSQL），都采用了这种设计原则，因此，扩展性都比较差

2.CP：也就是强调一致性（C）和分区容忍性（P），放弃可用性（A），当出现网络分区的情况时，受影响的服务需要等待数据一致，因此在等待期间就无法对外提供服务

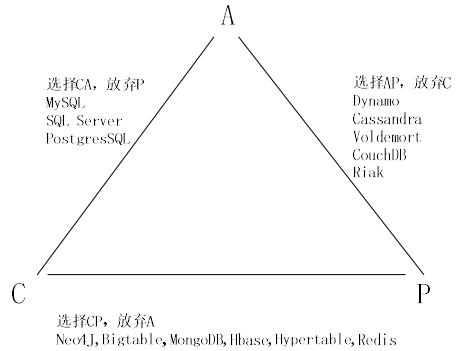
3.AP：也就是强调可用性（A）和分区容忍性（P），放弃一致性（C），允许系统返回不一致的数据

5.5.2 BASE

说起BASE（**Basically Availble, Soft-state, Eventual consistency**），不得不谈到ACID。

ACID	BASE
原子性(**A**tomicity)	基本可用(**B**asically **A**vailable)
一致性(**C**onsistency)	软状态/柔性事务(**S**oft state)
隔离性(**I**solation)	最终一致性 (**E**ventual consistency)
持久性 (**D**urable)	

5.5.1 CAP



不同产品在CAP理论下的不同设计原则

ACID

A（Atomicity）：原子性，是指事务必须是原子工作单元，对于其数据修改，要么全都执行，要么全都不执行

C（Consistency）：一致性，是指事务在完成时，必须使所有的数据都保持一致状态

I（Isolation）：隔离性，是指由并发事务所做的修改必须与任何其它并发事务所做的修改隔离

D（Durability）：持久性，是指事务完成之后，它对于系统的影响是永久性的，该修改即使出现致命的系统故障也将一直保持

BASE

BASE的基本含义是基本可用（Basically Availble）、软状态（Soft-state）和最终一致性（Eventual consistency）：

- **基本可用**

  基本可用，是指一个分布式系统的一部分发生问题变得不可用时，其他部分仍然可以正常使用，也就是允许分区失败的情形出现

- **软状态**

  “软状态（soft-state）”是与“硬状态（hard-state）”相对应的一种提法。数据库保存的数据是“硬状态”时，可以保证数据一致性，即保证数据一直是正确的。“软状态”是指状态可以有一段时间不同步，具有一定的滞后性

5.5.3 最终一致性

最终一致性根据更新数据后各进程访问到数据的时间和方式的不同，又可以区分为：

●**因果一致性：**如果进程A通知进程B它已更新了一个数据项，那么进程B的后续访问将获得A写入的最新值。而与进程A无因果关系的进程C的访问，仍然遵守一般的最终一致性规则

●**“读己之所写”一致性：**可以视为因果一致性的一个特例。当进程A自己执行一个更新操作之后，它自己总是可以访问到更新过的值，绝不会看到旧值

●**单调读一致性：**如果进程已经看到过数据对象的某个值，那么任何后续访问都不会返回在那个值之前的值

5.5.2 BASE

BASE的基本含义是基本可用（Basically Availble）、软状态（Soft-state）和最终一致性（Eventual consistency）：

- 最终一致性

一致性的类型包括强一致性和弱一致性，二者的主要区别在于高并发的数据访问操作下，后续操作是否能够获取最新的数据。

对于强一致性而言，当执行完一次更新操作后，后续的其他读操作就可以保证读到更新后的最新数据；反之，如果不能保证后续访问读到的都是更新后的最新数据，那么就是弱一致性。

最终一致性只不过是弱一致性的一种特例，允许后续的访问操作可以暂时读不到更新后的数据，但是经过一段时间之后，必须最终读到更新后的数据。

最常见的实现最终一致性的系统是DNS（域名系统）。一个域名更新操作根据配置的形式被分发出去，并结合有过期机制的缓存；最终所有的客户端可以看到最新的值。

6.5.3 最终一致性

最终一致性根据更新数据后各进程访问到数据的时间和方式的不同，又可以区分为：

●**会话一致性：**它把访问存储系统的进程放到会话（session）的上下文中，只要会话还存在，系统就保证“读己之所写”一致性。如果由于某些失败情形令会话终止，就要建立新的会话，而且系统保证不会延续到新的会话

●**单调写一致性：**系统保证来自同一个进程的写操作顺序执行。系统必须保证这种程度的一致性，否则就非常难以编程了

5.5.3 最终一致性

如何实现各种类型的一致性？

对于分布式数据系统：

•N — 数据复制的份数
•W — 更新数据时需要保证写完成的节点数
•R — 读取数据的时候需要读取的节点数

如果W+R>N，写的节点和读的节点重叠，则是强一致性。例如对于典型的一主一备同步复制的关系型数据库，N=2,W=2,R=1，则不管读的是主库还是备库的数据，都是一致的。一般设定是R＋W = N+1，这是保证强一致性的最小设定

如果W+R<=N，则是弱一致性。例如对于一主一备异步复制的关系型数据库，N=2,W=1,R=1，则如果读的是备库，就可能无法读取主库已经更新过的数据，所以是弱一致性。

6.6 从NoSQL到NewSQL数据库

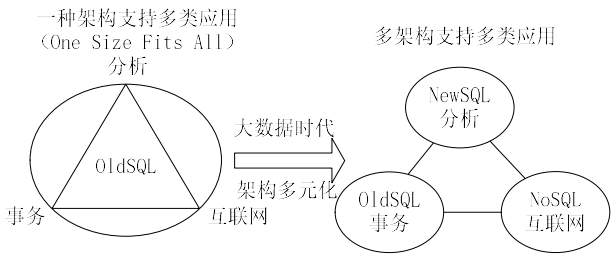


图6 大数据引发数据处理架构变革

5.5.3 最终一致性

对于分布式系统，为了保证高可用性，一般设置N>=3。不同的N,W,R组合，是在可用性和一致性之间取一个平衡，以适应不同的应用场景。

•如果N=W,R=1，任何一个写节点失效，都会导致写失败，因此可用性会降低，但是由于数据分布的N个节点是同步写入的，因此可以保证强一致性。

实例：HBase是借助其底层的HDFS来实现其数据冗余备份的。**HDFS采用的就是强一致性保证。**在数据没有完全同步到N个节点前，写操作是不会返回成功的。也就是说它的W＝N，而读操作只需要读到一个值即可，也就是说它R＝1。

•像Voldemort，Cassandra和Riak这些类Dynamo的系统，通常都允许用户按需要设置N，R，W三个值，即使是设置成W＋R<= N也是可以的。也就是说他允许用户在强一致性和最终一致性之间自由选择。而在用户选择了最终一致性，或者是W<N的强一致性时，则总会出现一段“各个节点数据不同步导致系统处理不一致的时间”。为了提供最终一致性的支持，这些系统会提供一些工具来使数据更新被最终同步到所有相关节点。

6.4 从NoSQL到NewSQL数据库

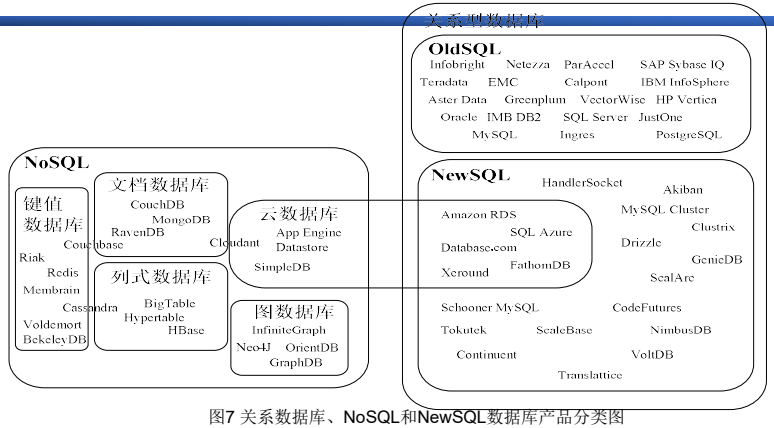


图7 关系数据库、NoSQL和NewSQL数据库产品分类图


Thank You!