南京大学 软件学院
NANJING UNIVERSITY · SOFTWARE INSTITUTE

数据管理基础

ch65 主要的NoSQL数据模型

Software Institute
Nanjing University
Bei Jia

键值数据模型与文档数据模型

- 这两类数据库都包含大量聚合，每个聚合中都有一个获取数据所用的键或ID。
- 两种模型的区别是:
  - 键值数据库的聚合不透明，只包含语义中立的大块信息
    - 数据库可能会限制聚合的总大小，但除此之外，聚合中可以存储任意数据。
    - 在键值数据库中，要访问聚合内容，只能通过键来查找
  - 在文档数据库的聚合中，可以看到其结构。
    - 限制其中存放的内容，它定义了其允许的结构与数据类型
    - 能够更加灵活地访问数据。通过用聚合中的字段查询，可以只获取一部分聚合，而不用获取全部内容
    - 可以按照聚合内容创建索引

列族存储 1

- 大部分数据库都以行为单元存储数据。然而，有些情况下写入操作执行得很少，但是经常需要一次读取若干行中的很多列。此时，列存储数据库将所有行的某一组列作为基本数据存储单元
- 列族数据库将列组织为列族。每一列都必须是某个列族的一部分，而且访问数据的单元也得是列
  - 某个列族中的数据经常需要一起访问
- 列族模型将其视为两级聚合结构(two-level aggregate structure)。
  - 第一个键通常代表行标识符，与“键值存储”相同，可以用它来获取想要的聚合
  - 列族结构与“键值存储”的区别在于，其“行聚合”(row aggregate)本身又是一个映射，其中包含一些更为详细的值。这些“二级值” (second-level value)就叫做“列”。与整体访问某行数据一样，我们也可以操作特定的列

列族存储 2

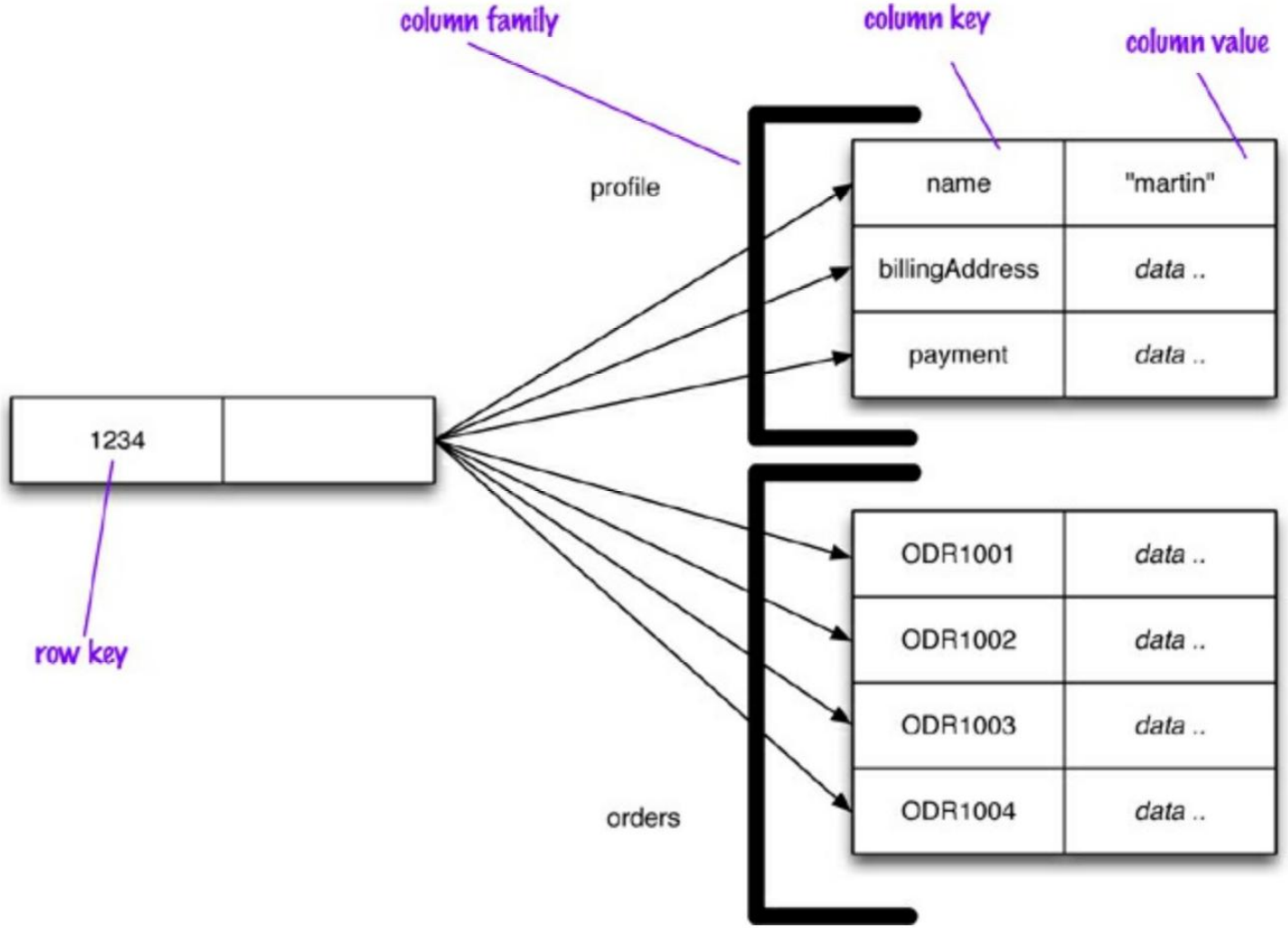

Figure 2.5. Representing customer information in a column-family structure

列族存储 3

- 两种数据组织方式
  - 面向行(row-oriented)：每一行都是一个聚合(例如ID为1234的顾客就是一个聚合)，该聚合内部存有一些包含有用数据块(客户信息、订单记录)的列族
  - 面向列(column-oriented): 每个列族都定义了一种记录类型(例如客户信息)，其中每行都表示一条记录。数据库中的大“行”理解为列族中每一个短行记录的串接

面向聚合的数据模型

- 共同点
  - 都使用聚合这一概念，而且聚合中都有一个可以查找其内容的索引键
  - 在集群上运行时，聚合是中心环节，因为数据库必须保证将聚合内的数据存放在同一个节点上
  - 聚合是“更新”操作的最小数据单位(atomic unit)，对事务控制来说，以聚合为操作单元
- 差别
  - 键值数据模型将聚合看作不透明的整体，只能根据键来查出整个聚合，而不能仅仅查询或获取其中的一部分
  - 文档模型的聚合对数据库透明，于是就可以只查询并获取其中一部分数据了，不过，由于文档没有模式，因此在想优化存储并获取聚合中的部分内容时，数据库不太好调整文档结构
  - 列族模型把聚合分为列族，让数据库将其视为行聚合内的一个数据单元。此类聚合的结构有某种限制，但是数据库可利用此种结构的优点来提高其易访问性。

图结构

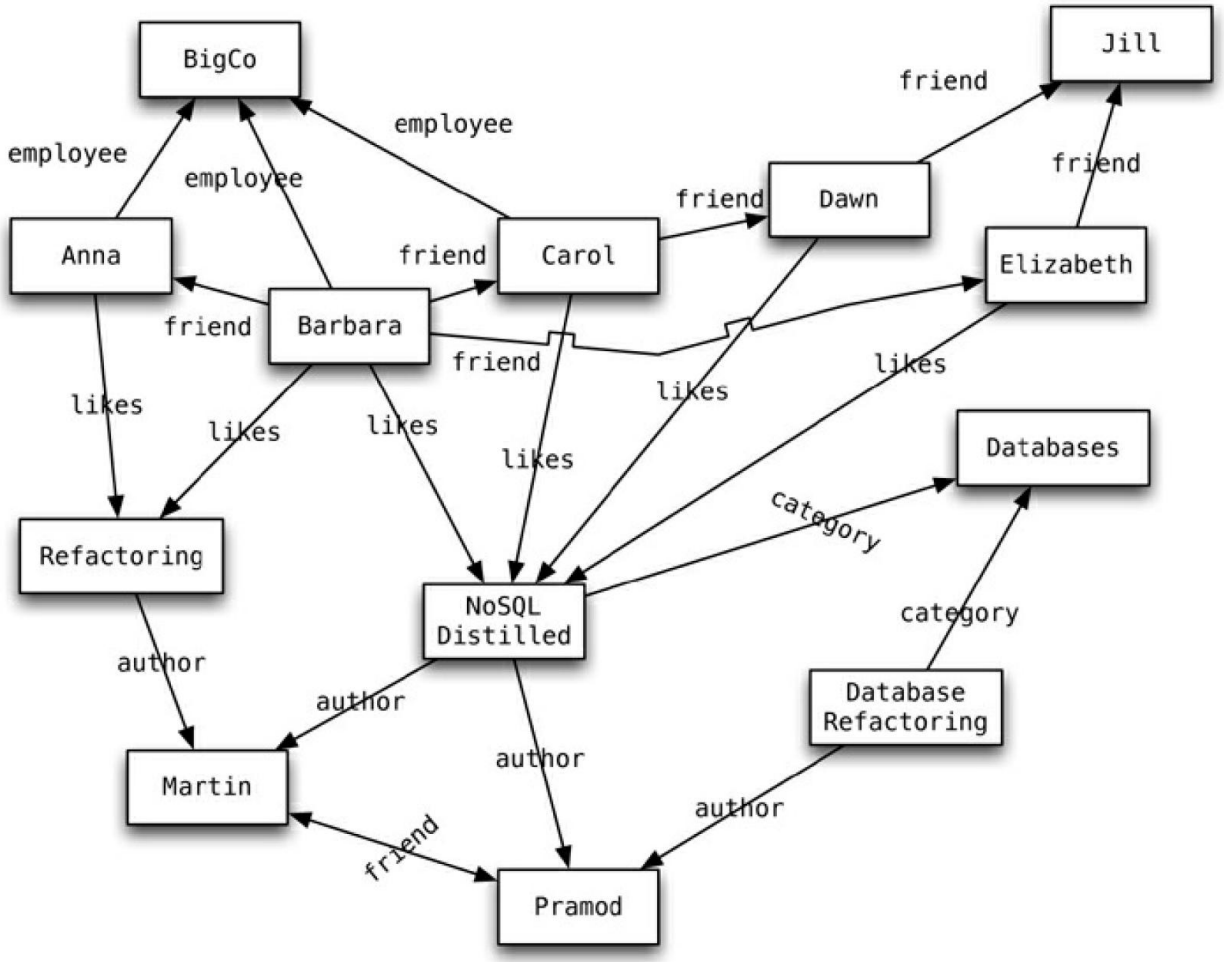

Figure 3.1. An example graph structure

图数据库

- 图数据库的基本数据模型：由边(或称“弧”，arc)连接而成的若干节点。
- 可以用专门为“图”而设计的查询操作来搜寻图数据库的网络了
  - 指定节点，通过边进行查询
- 关系型数据可以通过“外键”实现，查询中的多次连接，效率较差

无模式

- 关系型数据库中，首先必须定义“模式”，然后才能存放数据。
- NoSQL数据库，无模式：
  - “键值数据库”可以把任何数据存放在一个“键”的名下。
  - “文档数据库” 对所存储的文档结构没有限制
  - 在列族数据库中，任意列里面都可以随意存放数据
  - 图数据库中可以新增边，也可以随意向节点和边中添加属性。

格式不一致的数据

- 每条记录都拥有不同字段集(set of field)
- 关系型数据库中，“模式”会将表内每一行的数据类型强行统一，若不同行所存放的数据类型不同，那这么做就很别扭。
  - 要么得分别用很多列来存放这些数据，而且把用不到的字段值填成null(这就成了“稀疏表”，sparse table)，
  - 要么就要使用类似custom column 4这样没有意义的列类型。
- 无模式表则没有这么麻烦，每条记录只要包含其需要的数据即可，不用再担心上面的问题了。

无模式的问题

- 存在“隐含模式”。在编写数据操作代码时，对数据结构所做的一系列假设
  - 应用与数据的耦合问题
  - 无法在数据库层级优化和验证数据
- 在集成数据库中，很难解决
  - 使用应用程序数据库，并使用Web Services、SOA等完成集成
  - 在聚合中为不同应用程序明确划分出不同区域
    - 在文档数据库中，可以把文档分成不同的区段(section)
    - 在列族数据库，可以把不同的列族分给不同的应用程序

南京大学 软件学院
NANJING UNIVERSITY · SOFTWARE INSTITUTE

数据管理基础

ch68 放宽"一致性"和"持久性"约束

Software Institute
Nanjing University
Bei Jia

使用事务保障“一致性”

- 使用“事务”达成强一致性
- 引入放松“隔离级别”（isolation level)的功能，以允许查询操作读取尚未提交的数据。
  - 读未提交，一个事务可以读取另一个未提交事务的数据。脏读
  - 读已提交，一个事务要等另一个事务提交后才能读取数据。不可重复读
  - 可重复读，在开始读取数据（事务开启）时，不再允许修改操作。幻读
  - 可串行化，事务串行化顺序执行。严格一致性，效率是一个问题

事务的问题

- 在并发不大的前提下，是否需要事务
- 在数据较多的情况下，为了让应用性能符合用户要求，它们必须弃用“事务”
  - 尤其在需要引入分片机制时，更是如此
- 在分布式应用中，如事务的业务范围涉及多个以网络连接的参与者。其规模、复杂度和波动性均导致无法使用事务进行良好描述

CAP定理

- CAP定理:给定“一致性”(Consistency)、“可用性”(Availability)、“分区耐受性”(Partition tolerance) 这三个属性，我们只能同时满足其中两个属性。
  - “一致性”
  - “可用性”，如果客户可以同集群中的某个节点通信，那么该节点就必然能够处理读取及写入操作。
  - “分区耐受性”，如果发生通信故障，导致整个集群被分割成多个无法互相通信的分区时(这种情况也叫“ 脑裂”，split brain)，集群仍然可用。

“脑裂”的例子

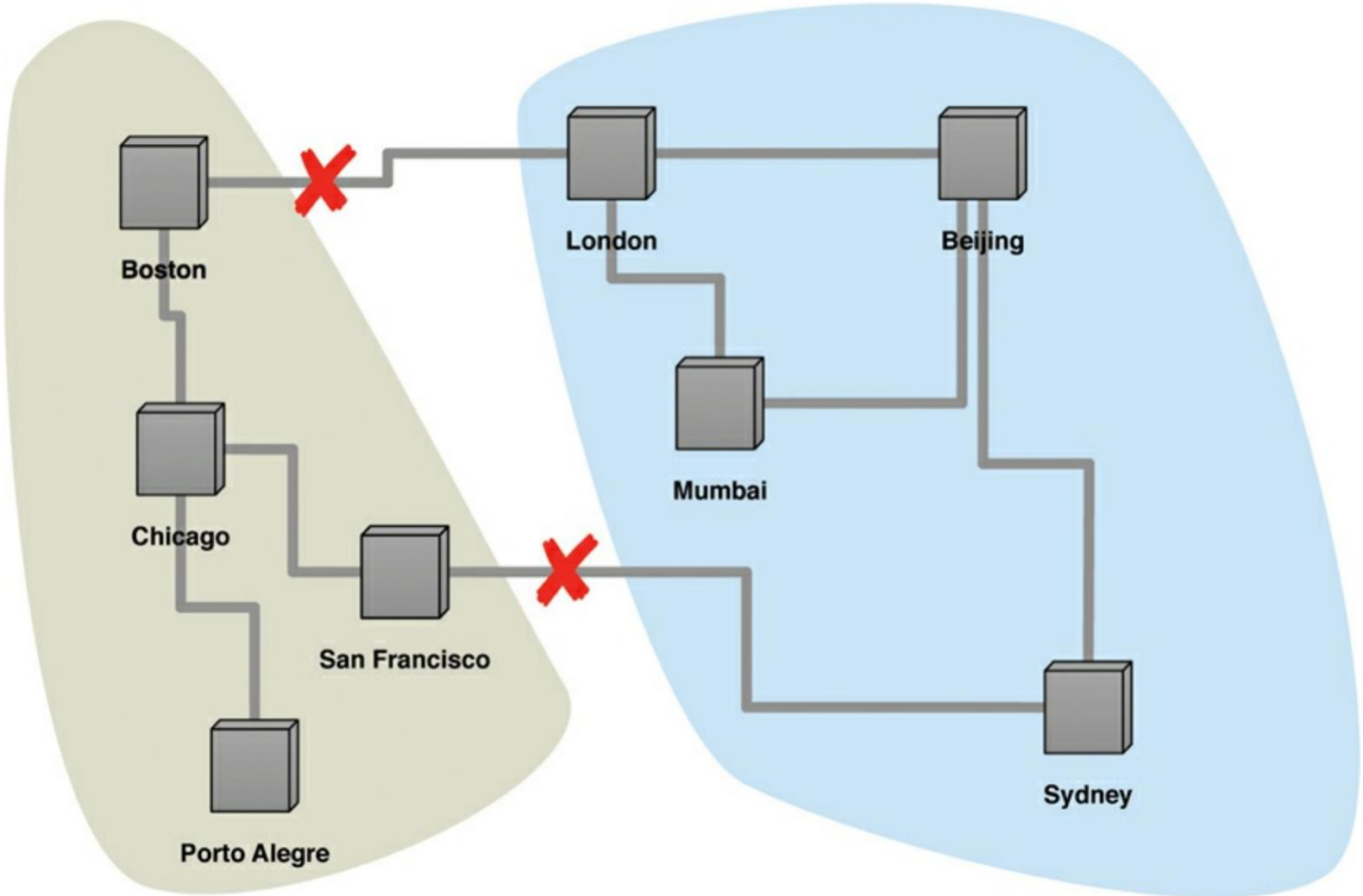

Figure 5.3. With two breaks in the communication lines, the network partitions into two groups.

CA系统

- CA系统，也就是具备“一致性”(Consistency)与“可用性”(Availability)， 但却不具备“分区耐受性”的系统
  - 大多数关系型数据库
- CA集群
  - 无法保证“分区耐受性”，这使得一旦“分区”发生，所有节点必须停止运作
  - CAP中的，可用性定义为“系统中某个无故障节点所接收的每一条请求，无论成功或失败，都必将得到响应。”
  - 介于此时所有节点均为故障节点，不违反CAP中的“可用性”

CAP定理的现实含义

- 尽管“CAP定理”经常表述为“三个属性中只能保有两个”，实际上当系统可能会遭遇“分区”状况时(比如分布式系统)，需要在“一致性”与“可用性”之间进行权衡。
  - 这并不是个二选一的决定，通常来说，我们都会略微舍弃“一致性”，以获取某种程度的“可用性”
  - 这样的系统，既不具备完美的“一致性”，也不具备完美的“可用性”
  - 但是能够满足需要

缺乏"可用性"的例子

- 假设Martin与Pramod都想预订某旅馆的最后一间客房，预订系统使用"对等式分布模型"，它由两个节点组成
  - Martin 使用位于伦敦的节点，而Pramod使用位于孟买的节点。
  - 若要确保一致性，那么当Martin要通过位于伦敦的节点预订房间时，该节点在确认预订操作之前，必须先告知位于孟买的节点。
  - 两个节点必须按照相互一致的顺序来处理它们所收到的操作请求
- 此方案保证了"一致性"，但是假如网络连接发生故障，那么由故障导致的两个"分区"系统，就都无法预订旅馆房间了，于是系统失去了"可用性"。

改善"可用性"的例子

- 指派其中一个节点作为某家旅馆的"主节点"，确保所有预订操作都由"主节点"来处理。
  - 假设位于孟买的节点是"主节点"，那么在两个节点之间的网络连接发生故障之后，它仍然可以处理该旅馆的房间预订工作，这样Pramod将会订到最后一间客房
  - 位于伦敦的用户看到的房间剩余情况会与孟买不一致，但是他们无法预订客房，于是就出现了"更新不一致"现象。
  - Martin可以和位于伦敦的节点通信，但是该节点却无法更新数据。于是出现了"可用性"故障(availability failure)
- 这种在"一致性"与"可用性"之间所做的权衡，能正确处理上述特殊状况。

进一步改善“可用性”的例子

- 让两个“分区”系统都接受客房预订请求，即使在发生网络故障时也如此。
  - 这种方案的风险是，Martin和Pramod有可能都订到了最后一间客房。然而，根据这家旅馆的具体运营情况，这也许不会出问题：
    - 通常来说，旅行公司都允许一定数量的超额预订，这样的话，如果有某些客人预订了房间而最终没有人住，那么就可以把这部分空余房间分给那些超额预订的人了
    - 与之相对，某些旅馆总是会在全部订满的名额之外多留出几间客房，这样万一哪间客房出了问题，或者在房间订满之后又来了一位贵宾，那么旅馆可以把客人安排到预留出来的空房中
    - 还有些旅馆甚至选择在发现预订冲突之后向客户致歉并取消此预订。
- 该方案所付出的代价，要比因为网络故障而彻底无法预订的代价小。

一个写入不一致的例子

- 购物车是允许“写入不一致”现象的一个经典示例
  - 即使网络有故障，也总是能够修改购物车中的商品。
  - 这么做有可能导致多个购物车出现
  - 而结账过程则会将两个购物车合并，具体做法是，将两个购物车中的每件商品都拿出来，放到另外一个购物车中，并按照新的购物车结账。
  - 这个办法基本上不会出错，万一有问题，客户也有机会在下单之前先检视一下购物车中的东西。

BASE

- 与关系型数据库所支持的ACID事务不同，NoSQL系统具备“BASE属性”
  - 基本可用，Basically Available
  - 柔性状态，Soft state
  - 最终一致性，Eventual consistency
- “ACID”与“BASE"不是非此即彼的关系，两者之间存在着多个逐渐过渡的权衡方案可选。

“一致性”与“延迟”之间取舍

- 在权衡分布式数据库的“一致性”时，与其考虑如何权衡“一致性”与“可用性”，不如思考怎样在“一致性”与“延迟”(latency)之间取舍。
  - 参与交互操作的节点越多，“一致性”就越好
  - 然而，每新增一个节点，都会使交互操作的响应时间变长
  - “可用性”可以视为能够忍受的最大延迟时间，一旦延迟过高，我们就放弃操作，并认为数据不可用
  - 这样一来，就和“CAP定理”对“可用性”所下的定义相当吻合了

“持久性”的权衡

- 数据库大部分时间都在内存中运行，更新操作也直接写入内存，并且定期将数据变更写回磁盘
  - 可以大大提高响应请求的速度。
  - 代价在于，一旦服务器发生故障，任何尚未写回磁盘的更新数据都将丢失。
- 多用户的“会话状态”信息
  - 会话数据就算丢失，与应用系统效率相比，也不过是个小麻烦。这时可以考虑非持久性写入操作”(nondurable write)。
  - 可以在每次发出请求时，指定该请求所需的持久性。从而，把某些极为重要的更新操作立刻写回磁盘。
- 捕获物理设备的遥测数据(telemetric data)。就算最近的更新数据可能会因为服务器发生故障而丢失，也还是选择把快速捕获数据放在首位。

分布模型中"持久性"的权衡 1

- 如一个节点处理完更新操作之后，在更新数据尚未复制到其他节点之前就出错了，那么则会发生"复制持久性"(replication durability) 故障。
- 假设有一个采用"主从式分布模型"的数据库，在其主节点出错时，它会自动指派一个从节点作为新的主节点。
  - 若主节点发生故障，则所有还未复制到其他副本的写入操作就都将丢失
  - 一旦主节点从故障中恢复过来，那么，该节点上的更新数据就会和发生故障这段时间内新产生的那些更新数据相冲突
  - 我们把这视为一个"持久化"问题，因为主节点既然已经接纳了这个更新操作，那么用户自然就会认为该操作已经顺利执行完，但实际上，这份更新数据却因为主节点出错而丢失了

分布模型中"持久性"的权衡 2

- 解决方案：
  - 不重新指派新的主节点
    - 在主节点出错之后迅速将其恢复
  - 确保主节点在收到某些副本对更新数据的确认之后，再告知用户它已接纳此更新
    - 从节点发生故障时，集群不可用
    - 拖慢更新速度
- 与处理"持久性"的基本手段类似，也可以针对单个请求来指定其所需的持久性。

南京大学 软件学院
NANJING UNIVERSITY · SOFTWARE INSTITUTE

数据管理基础

ch69 仲裁

Software Institute
Nanjing University
Bei Jia

写入仲裁

- 处理请求所的节点越多，避免“不一致”问题的能力就越强，要想保“强一致性”(strong consistency)， 需要使用多少个节点才行?
- “对等式分布模型”：
  - “写入仲裁”(write quorum)：如果发生两个相互冲突的写入操作，那么只有其中一个操作能为超过半数的节点所认可，W>N/2 。即，参与写入操作的节点数(W)，必须超过副本节点数(N)的一半。副本个数又称为“复制因子”
- “主从式分布模型”
  - 只需要向主节点中写入数据

读取仲裁

- 想要保证能够读到最新数据，必须与多少个节点联系才行?
- “对等式分布模型”:
  - 只有当R+W>N时，才能保证读取操作的“强一致性”。其中，执行读取操作时所需联系的节点数(R)，确认写入操作时所需征询的节点数(W)，以及复制因子(N)
- “主从式分布模型”
  - 只需从主节点中读取数据

复制因子

- “复制因子”(replication factor)。
  - 一个集群有100 个节点，然而其“复制因子”可能仅仅是3，因为大部分数据都分布在各个"分片”之中。
- 将“复制因子”设为3，就可以获得足够好的“故障恢复能力”了。
  - 如果只有一个节点出错，那么仍然能够满足读取与写入所需的最小法定节点数。
  - 若是有自动均衡(automatic rebalancing) 机制，那么用不了多久，集群中就会建立起第三个副本，在替代副本建立好之前，再次发生副本故障的概率很小。

实际情况

- 需要在“一致性”与“可用性”之间权衡，参与某个操作的节点数，可能会随着该操作的具体情况而改变。
  - 在写入数据时，根据“一致性”与“可用性”这两个因素的重要程度，有一些更新操作可能需要获得足够的节点支持率才能执行，而另外一些则不需要。
  - 与之相似，某些读取操作可能更看中执行速度，而可以容忍过时数据，此时，它就可以少联系几个节点。
- 通常需要协调考虑读、写两种情况：
  - 假设需要快速且具备“强一致性”的读取操作，那么写入操作就要得到全部节点的确认才行，这样的话，只需联系一个节点，就能完成读取操作了(N=3，W=3，R=1)
  - 但是，这个方案意味着，写入操作会比较慢，因为它们必须得到全部三个节点确认之后，才能执行，而且此时连一个节点都不能出错。