文 复制、分片和路由 (/a/1190000004485355)

**xixicat** (https://segmentfault.com/u/xixicat) 5 天前发布

> 分布式理论系列

- 从ACID到CAP到BASE (https://segmentfault.com/a/1190000004468442)
- 2PC到3PC到Paxos到Raft到ISR (https://segmentfault.com/a/1190000004474543)
- 复制、分片和路由 (https://segmentfault.com/a/1190000004485355)
- 副本更新策略 (https://segmentfault.com/a/1190000004480546)
- 负载均衡算法及手段 (https://segmentfault.com/a/1190000004492447)

## 序

本文主要讲述分布式nosql的两大特性：复制和分片。传统数据库采用纵向Scale Up的方式，即改善单机硬件资源配置来解决问题；主流大数据存储与计算系统采用横向Scale Out的方式，支持系统可扩展性，即通过增加机器来获得水平扩展能力。

对于海量数据，通过数据分片（shard/partition）来讲数据进行切分并分配到各个机器中去，数据分片后，如何能够找到某条记录的存储位置就成为必然要解决的问题，这一般称为数据路由（Routing）。

对于海量数据，通过数据分片实现系统的水平扩展，通过数据复制保证数据的高可用性。

数据复制除了可保证可用性之外，还可以增加读操作的效率，即客户端可以从多个备份数据中选择物理距离较近的进行读取，这既增加了读操作的并发性又可以提高单次读的读取效率。

分片与复制可以组合，即同时采用主从复制与分片，则系统有多个节点，对每项数据来说，负责它的主节点只有一个。

## 聚合

NoSQL本质上是面向聚合操作的，也就是它将一组相互关联的对象视为一个整体单元来操作，这个单元就叫做聚合。

聚合是相对于元组来说的，元组不能在元组中嵌套另外一个元组，也不能包含由值或元组组成的列表，这是传统关系型数据库的规范。其优势是在于消除数据的冗余和一致性。但是对于join操作，复杂了的话，就很无能为力。

聚合的话，一方面是可以很方面的解决join操作问题的，但是其弱点是对于检索记录（不按聚合维度检索），稍微欠缺。

## 数据分布的两条路径

1、复制（replication）：将同一份数据拷贝到多个节点（主从master-slave方式、对等式peer-to-peer）
2、分片（sharding）：将不同数据存放在不同节点

如果想增加系统的读取性能，复制，增加slave节点即可；
如果想提升写入性能，则对数据进行分片。

## 分片

一般来说，数据库的繁忙体现在：不同用户需要访问数据集中的不同部分，这种情况下，我们把数据的各个部分存放在不同的服务器/节点中，每个服务器/节点负责自身数据的读取与写入操作，以此实现横向扩展，这种技术成为分片，即sharding。

理想情况下，不同的节点服务于不同的用户，每个用户只需要与一个节点通信，并且很快就能获得服务器的响应。当然理想情况比较罕见，为了获得近乎理想的效果，必须保证需要同时访问的那些数据都存放在同一个节点上，而且节点必须排布好这些数据块，使得访问速度最优。

为此，必须考虑：

- 1）怎样存放数据，才能保证用户基本上只需要从一个节点获取它。如果使用的是面向聚合的数据库而非面向元组的数据库，那么就非常容易解决了。之所以设计聚合这一结构，就是为了把那些经常需要同时访问的数据存放在一起。因此，可以把聚合作为分布数据的单元。
- 2）另外还要考虑的是：如何保持负载均衡。即如何把聚合数据均匀地分布在各个节点中，让它们需要处理的负载量相等。负载分布情况可能随着时间变化，因此需要一些领域特定的规则。比如有的需要按字典顺序，有的需要按逆域名序列等。

很多NoSQL都提供自动分片（auto-sharding）功能，可以让数据库自己负责把数据分布到各个分片，并且将数据访问请求引导到适当的分片上。

分片可以极大地提高读取性能，但对于要频繁写的应用，帮助不大。另外，分片对改善故障恢复能力并没有帮助，但是它减少了故障范围，只有访问这个节点的那些用户才会受影响，其余用户可以正常访问。虽然数据库缺失了一部分，但是还是其余部分还是可以正常运转。

## 路由

## 路由的两级映射抽象模型

1）Key-Partition映射，数据记录到分片的映射（多对一）
2）Partition-Machine映射，分片到物理机器的映射（多对一）

## 分片类型：

### 1）哈希分片，点查询，采用哈希函数建立Key-Partition映射（大多数KV数据库都支持此方式）

通过哈希函数来进行数据分片，主要有Round Robbin、虚拟桶、一致性哈希三种算法。

- A、Round Robbin
  俗称哈希取模算法，H(key) = hash(key) mode K（其中对物理机进行从0到K-1编号，key为某个记录的主键，H（key）为存储该数据的物理机编号）。好处是简单，缺点是增减机器要重新hash，缺乏灵活性。它实际上是将物理机和数据分片两个功能点合二为一了，因而缺乏灵活性。
- B、虚拟桶
  membase在待存储记录和物理机之间引入了虚拟桶，形成两级映射。其中key-partition映射采用哈希函数，partition-machine采用表格管理实现。新加入机器时，只需要将原来一些虚拟桶划分给新的机器，只要修改partition-machine映射即可，具有灵活性。
- C、一致性哈希
  一致性哈希是分布式哈希表的一种实现算法，将哈希数值空间按照大小组成一个首尾相接的环状序列，对于每台机器，可以根据IP和端口号经过哈希函数映射到哈希数值空间内。通过有向环顺序查找或路由表（Finger Table）来查找。对于一致性哈希可能造成的各个节点负载不均衡的情况，可以采用虚拟节点的方式来解决。一个物理机节点虚拟成若干虚拟节点，映射到环状结构的不同位置。

### 2）范围分片，范围查询，BigTable、Azure采用此方式，也可以支持点查询

# 复制

## 主从复制

master-slave模式，其中有个master节点，存放权威数据，通常负责数据的更新，其余节点都叫做slave节点，复制操作就是让slave节点的数据与master节点的数据同步。
好处是：
1）在需要频繁读取的情况下，有助于提升数据的访问（读取从库分担压力），还可以增加多个slave节点进行水平扩展，同时处理更多的读取请求，但是对于写操作频繁的场景，则没有什么帮助
2）可以增强读取操作的故障恢复能力。万一一个slave出故障，还有其他slave支撑访问。

问题：
数据一致性，如果数据更新没有全部通知到slave节点，则会导致数据不一致。

## 对等复制

主从复制有助于增强读取操作的故障恢复能力，然而对写操作没有帮助。它所提供的故障恢复能力，只有在从节点出错时才能体现出来，master仍然是系统的瓶颈和弱点。

对等复制，是指两个节点相互为各自的副本，也同时可以接受写入请求，丢失其中一个不影响整个数据库的访问。

但是，同时接受写入请求，容易出现数据不一致问题，实际使用上，通常是只有一个节点接受写入请求，另一个master作为stand-by，在对方挂掉的时候自动承接写操作请求，独当一面。

# 参考


- 大数据系列 (一)、数据分片与路由(Hash partition and Routing) (http://blog.csdn.net/gdhuyufei/article/details/42101231)



5 天前发布 (/a/1190000004485355)

xixicat (https://segmentfault.com/blog/xixicat)

spring boot , docker and so on

xixicat (https://segmentfault.com/u/xixicat)
作者

关注专栏

分享扩散：