数据仓库介绍

1. 数据仓库是用来存储数据的，可以根据存储数据类型的不同分为以下几种
    - 数据库：储存实际需要处理或者处理完的大规模数据，如mongo， mysql， hbase， redis
    - 队列：同样储存大量的数据，但是一般作为缓存，中间件，消费完就会删除， 如果rabbitmq，kafka
    - 配置中心：存储分布式集群中的元数据，控制信息等，如： zookeeper， etcd，

数据库

1. 数据库主要分为关系型数据库（mysql， oracle）， key-value（redis， memcached），文档数据库（mongo， Elasticsearch），列数据库（hbase等，不是很熟）
2. 下面主要针对mysql， mongo， redis， Elasticsearch， kafka等为代表进行分析分布式仓库的一些设计理念和原理

分布式架构

在数据仓库中，有些框架本生设计时就是分布式的（kafka， Elasticsearch），有些则是通过分布式的部署来实现分布式的特性（mysql，Elasticsearch）。分布式的架构主要分为两种：

1. 主从模式（master-slave）
2. 分片副本模式（shard-replica）

主从架构

主从模式是最简单的架构，所有的数据都写在一个master服务器中。主从的原因主要有两个：一是读写分离，提高查询的速度以及减少master的性能瓶颈；二是高可用，防止master出现问题的时候可以快速却换到slave机器上，保证服务的继续运行。

主从架构用得比较多的感觉是mysql，mysql可以通过binlog的方式让slave跟master执行同样的数据操作，最终保证slave与master的数据一致。

主从架构带来的问题：当master出现宕机时，一般需要人工干预，手工将数据的连接地址由master改为slave。如果是分片副本模型，一般会自动选取新的leader，然后继续提供服务，期间不需要人工干预。

分片副本

1. 相对于主从架构来说，分片副本模式比较复杂，需要更多的逻辑来保证数据的均匀分布，一致性，leader的选举等功能。分片副本架构可以分为以下几个主要方面：
    - 如何分片
    - 查询时怎么获取数据
    - 分片如何复制数据
    - leader选举

如何分片

1. 分片其实就是将数据均匀的分布到各个机器上。最常见和使用最广的方法莫过于 hash了，用户指定需要hash的字段，数据仓库服务将数据按照指定字段进行hash，然后分布到各个机器上。但是根据不同的应用场景，我们需要使用不同的分布算法，下面是最常用的3种方法。

    - hash 算法
    - range分片算法
    - 一致性hash算法（consistent hashing）

2. hash 算法
   hash算法前面已经有提到，其实就是 `y = hash(x)` ，那在实际数据仓库中都是怎么具体实现的呢。这里介绍一个redis的实现过程（http://redis.io/topics/partitioning），别的框架大同小异。

```
1. assume key=foobar; num=4(redis实例个数数量)
2. val = crc32(foobar)=93024922
3. r = val%4 = 2(foobar存储到编号为2的实例上)
```

3. range 分片算法

    - 首先需要知道的是，mongo默认使用的分片算法居然是 range分片算法(https://docs.mongodb.com/manual/core/ranged-sharding/#ranged-sharding)。range算法是根据指定的key的大小范围进行映射数据到相应实例的。

    - 比如，有4个mongo实例用于存储用户数据，那么可以根据用户的userid作为range分片算法的关键key（mongo中叫做：range shard key），当有40个用户，userid分别为：1,2,3....40。那么前10个用户数据（1-10）存储在mongo1实例，11-20的用户在mongo2实例，21-30的用户在mongo3实例，最后31-40在mongo4实例。

    - range分片算法对于如何选择key也是很重要，直接影响到数据读写性能的高低，mongo中选择range shard key的标准如下，别的储存架构也可以参照一下

        - Large Shard Key Cardinality，key的范围要广，比如只有4个用户的数据，但是每个用户的关联数据量非常大，这时你如果用userid来分片，最多也只能将数据分配到4个实例上，就算有再多的实例也用不上了。
        - Low Shard Key Frequency：key中相同值要尽可能少，如果key里面大部分数据相同的话，会导致相同某个分片数据量非常大，但是别的分片数据非常少，负载不均衡。
        - Non-Monotonically Changing Shard Keys：key的值具有非单调性。为什么，如果你的key具有单调性如单调递增或者单调递减，也就是说每次收到的数据中key的值都在逐渐增加，或者逐渐减少。当第一个数据进来时，key的值为15且单调递增，此时可能放在第三个shard上，那么以后来的数据都只会从第三个shard开始写数据。

    - range分片算法的优点：由于range分片算法的特性知道，相邻的数据总是在相同的分片，所以当我们查询的条件总是搜索相邻某一部分数据时，此时的查询效率就非高

4. 一致性hash算法（consistent hashing）
   一致性hash平时接触得比较少，但是一致性hash算法还是非常重要的。思考这样一个场景，我们在redis集群中的4个实例使用 hash算法对数据进行分片储存，当数据量猛增时，我们需要增加一个redis的实例，这样看似没什么问题。但仔细一下，我们数据存储时是不是有问题呢，比如：数据data1之前通过hash函数 `y=hash(data1, 4)` 映射到了redis1这个实例上，现在由于实例变成5个了，那么取数据的时候使用hash函数 `y=hash(data1, 5)` ，肯定取到的是错误的值。同理，减少redis实例的时候也会出现存储数据出问题的情况。这时候就需要使用一致性hash算法来解决上述问题了。

- 一致性hash（consistent hashing）要解决的问题就是：当服务的实例数增加或者减少时不会影响之前数据的存取。
- 算法主要原理（http://blog.csdn.net/cywosp/article/details/23397179/）：
    - 预设一个环形的hash空间。（数据不再直接通过hash函数映射到机器上，而是将数据和机器分别映射到环形hash空间中。）
    - 数据通过hash函数映射到该环形hash空间上，
    - 机器以同样的hash函数映射到该环形hash空间上。
    - 环形hash空间上的数据往顺时针方向查找，查找到的第一个机器，则将该数据存储到找到的机器上。
    - 为了让数据均匀的分布在每台机器上，就需要让机器尽可能的均匀分布在环形hash空间，一般会采用虚拟节点的方式，将每一个实际机器虚拟成好几个。当总结点（所有虚拟节点）个数较多时，就可以认为这些节点在环形hash空间均匀分布。
- redis的文档中有提到使用hash一致性的算法，但是mongo的所有文档中并没有提到这个算法（只有hash算法和range分布算法）。
- 在mongo集群中使用分片副本架构，那么可以想象，集群部署时没有指定副本数（即没有副本），只要集群中有一个实例宕机了就会导致该实例上的数据没法使用，而且会导致之后有一部分的数据映射到该实例上也无法使用。

5. mysql的分片副本架构
   mysql一般使用主从比较多，但也可以部署为分片副本模式，就是常说的分库分表。其主要目的是为突破单节点数据库服务器的 I/O 能力限制，解决数据库扩展性问题。一般我们可以在应用层通过上述分片的原理实现分库分表的功能。
    - 垂直切分：将一个数据库分成好几个数据库
    - 水平切分：将同一张表中的数据通过hash路由到各个数据库实例中去
    - 缺点：
        - join表时很麻烦，无法通过一次操作完成，当多个实例获取完数据后还需要进一步处理结果。
          -

查询数据

分片就是将数据进行分开存储，那么查询的时候该怎么进行数据查询呢。这里结合Elasticsearch和Mongo的查询模式进行对比分析下，在两者的分片副本集群模式下，该查询方式的专业名称为：distributed search。Elasticsearch查询的过程分为2步：query then fetch，而mongo其实只有query，并没有fetch。下面分别介绍。

1. 协调节点：mongos 和 Elasticsearch node
    - 在集群模式下，每一个查询请求都是先到一个协调节点，这个节点负责将查询请求分发到每个主分片或者副本上，然后将每个分片返回的结果进行整合最后返回给客户端。这个协调点在mongo中是集群架构中的mongos，在Elasticsearch中则是coordinating node，它们的区别在于，mongos是固定的，但是Elasticsearch中的coordinating node则可以是集群中的任意一个结点，原因主要在Elasticsearch中每个结点之间是对等的，都存储了整个集群中的meta data。
    - 在Mongo中，整个集群在架构设计上就显示的进行了区分：mongs + config server + mongodb，mongs作为路由器，负责整个集群的数据存取规则，而config server（其实也是

一个mongod）则是存储集群中的meta data，包括路由和分片的配置等，mongs第一次启动时会去config server读取配置到内存中，自己本地并不会存储，架构中的mongodb则是存储实际数据的节点，包括shard和replica。

- 由于Mongo和Elasticsearch的集群架构的模式不同，所以路由规则也不同。Elasticsearch中每个Node都是一个Elasticsearch的实例，除了数据存储同时也包含路由功能，而且每个Elasticsearch实例本身包含多个分片和副本，而mongo中每个mongo服务的实例本身就是一个分片或者副本。这里不展开讨论了。

2. query

- 分布式查询的第一步就是query。mongo和Elasticsearch这一步基本相同，coordinating node（mongs/Elasticsearch node）负责将查询请求分发到每个存储节点上，存储节点将匹配的相关数据返回。但是返回的数据内容有些差别，mongodb返回的是查询到的每条完整数据，而Elasticsearch中返回的仅仅是每个文档的ID和sort value（文档相关度的评分等）。

- mongo中建议使用shard key进行分片，查询的时候同时带上这个shard key，这样mongs就能将请求分发到一个指定的shard上，而这个shard返回的数据可以直接返回给请求者，这样查询效率就非常高；如果查询请求中没有带上shard key，那么mongos需要将请求分发到每一个节点上，同时还需要将每个节点返回的结果进行合并处理，比如排序，分页等。所以当数据量非常大时，查询的时候如果没有带上shard key，那么会导致mongos的内存和cpu使用率爆炸，性能出现问题。

- Elasticsearch中能将每一个请求都转发到一个固定的shard上，这样查询的速度和效率是最高的。在Elasticsearch中，文档的_id就是一种默认的shard key，当通过 `index/_id/type` 进行请求时，Elasticsearch就能通过 `_id` 定位到一个确定的分片上的一个确定文档。当Elasticsearch也支持用户自定义路由，在插入文档和查询文档时，在请求中当 `routing` 关键字来进行路由，这样速度也会很快。Elasticsearch的查询过程可以分为以下三步：

  - coordinating node就请求分发到每一个节点上
  - 每个节点根据

3. fetch

- 为什么Elasticsearch还有一个fetch的步骤呢。前面我们提到，Elasticsearch的query阶段每个节点返回的数据都是文档id和sort value（文档相关性评分）。coordinating node得到每个节点返回的数据后，还需要根据文档相关性分数的高低进一步排序，然后将需要返回给用户的top N条文档，根据id从各个节点和分片中请求回来，最终统一返回给用户，这就是fetch的过程。

- 从百度和google的页面我们知道，搜索引擎都有进行分页，其实Elasticsearch也是默认分页的，如果请求没有指定分页的条数则默认返回前10条。当每个节点收到coordinating node的请求时，不管是查询还是聚合，每个节点都是进行top N的查询和聚合。在聚合时存在一个精确度的问题。

数据高可用

1. 日志复制

2. leader选举

3. 安全性约束