讨论课 2

1352911 曹琦

随着互联网时代的到来，计算机要管理的数据量呈指数级别地飞速上涨，如果在这爆发的关键时刻，系统不稳定或无法访问，那么对于业务将会是毁灭性的打击。

伴随着对于系统性能、成本以及扩展性的新需要，单机存储往往会遇到性能和单点故障问题，分布式系统应运而生。**本文主要介绍分布式数据存储，以案例形式对其导致的扩展性，负载均衡等问题进行探讨。**

分布式系统

分布式系统特点

1. 可扩展性

企业级应用需求经常随时间而不断变化，这也对企业级应用平台提出了很高的要求。企业级应用平台必须要能适应需求的变化，即具有可扩展性。分布式系统可以通过增加服务器数量来增强分布式系统整体的处理能力，以应对企业的业务增长带来的计算需求。

2. 廉价高效

由成本低廉的PC服务器组成的集群，在性能方面能够达到或超越大型机的处理性能，在成本上远低于大型机。这也是分布式系统最吸引人之处。成本低廉的PC服务器在硬件可靠性方面比大型机相去甚远，于是分布式系统由软件来对硬件进行容错，通过软件来保证整体系统的高可靠性。

3. 弹性扩展

应用服务层面的弹性扩展是随着业务量的涨落，分布式系统后台应用服务的实例能动态变化很方便地调度应用服务实例，从几十个到几百个甚至上千个，真正实现应用服务的弹性扩展。该特性对电商平台应对双十一等流量激增的风险尤为重要。

分布式系统类型

1. 分布式文件系统

分布式文件系统是分布式的基石，通常作为上层系统的底层存储。总体上看，分布式文件系统存储三种类型的数据: Blob 对象、定长块以及大文件。典型的系统有 Facebook Haystack 以及 Taobao File System(TFS)。

2. 分布式键值系统

分布式键值系统用于存储关系简单的半结构化数据，它只提供基于主键的 CRUD (Create /Read/Update/Delete) 功能。从数据结构的角度看，分布式键值系统与传统的哈

希表比较类似，不同的是，分布式键值系统支持将数据分布到集群中的多个存储节点。一般用作缓存，比如淘宝 Tair 以及 Memcache。一致性哈希是分布式键值系统中常用的数据分布技术。

3. 分布式表格系统

分布式表格系统用于存储关系较为复杂的半结构化数据。分布式表格系统以表格为单位组织数据，每个表格包括很多行，通过主键标识一行，同一个表格的多个数据行也不要求包含相同类型的列，支持根据主键的 CRUD 功能以及范围查找功能。 典型的系统包括 Google Bigtable 以及 Megastore，Microsoft Azure Table Storage，Amazon DynamoDB 等。

4. 分布式数据库

分布式数据库一般用于存储结构化数据。典型的系统包括 MySQL 数据库分片(MySQL Sharding)集群，Amazon RDS 以及Microsoft SQL Azure。分布式数据库支持的功能最为丰富，符合用户使用习惯，但可扩展性往往受到限制。当然，这一点并不是绝对的。Google Spanner 的扩展性就达到了全球级，它不仅支持丰富的关系数据库功能，还能扩展到多个数据中心的成千上万台机器。除此之外，阿里巴巴 OceanBase 也是一个支持自动扩展的分布式关系数据库。

数据存储

该部分主要探究的分布式数据存储方式是分布式数据库存储。

数据库的分表

1. 垂直拆分

垂直拆分是按功能模块拆分，多个数据库之间的表结构不同。垂直拆分实现简单，应用类型合适这种拆分方式时，能很好的起到分散数据库压力的作用。以豆瓣为例， 因为豆瓣的各核心业务/模块（书籍、电影、音乐）相对独立，数据的增加速度也比较平稳，采用垂直拆分较为合适。

2. 水平拆分

水平拆分是将同一个表的数据进行分块保存到不同的数据库中，这些数据库中的表结构完全相同。针对压力集中在特定功能模块的数据存储，比较适合水平拆分，比如又拍网和美丽说。又拍网时一个照片分享社区，核心业务对象是用户上传的照片，而照片数据的增加速度随着用户量的增加越来越快。压力基本上都在照片表上。美丽说是一个女性购物网站，同样压力较为集中，适合采用水平拆分。

水平分表的数据划分规则包括：

- 经常被一起请求的数据放入同一个shard
- 读写负载均衡
- 扩展（二次shard）时数据迁移方便按索引/映射表对应

比较常用的方法包括：

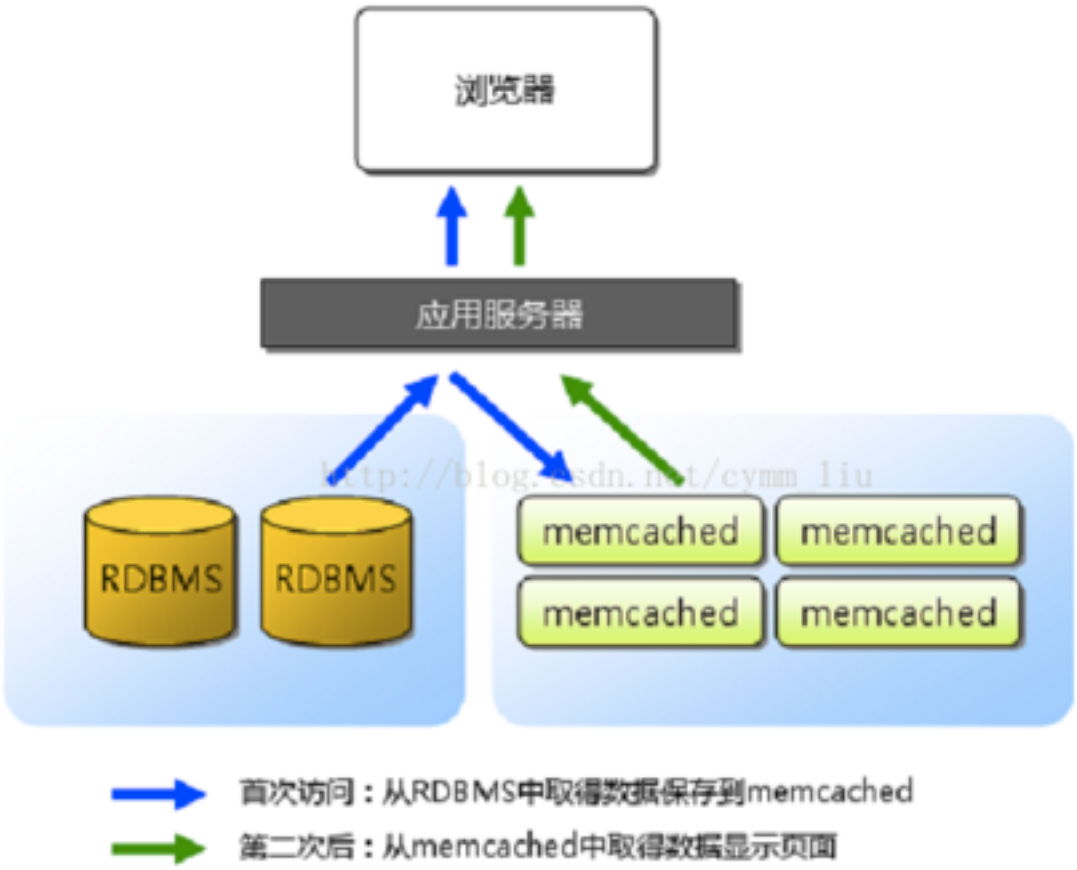
- 按算法拆分对应：方便高效，但是不能满足后续的伸缩性要求。

- 按索引/映射表拆分对应：灵活，有很好的伸缩性，性能略差于前一种方法。目前，美丽说和又拍网均采用该种方法对应。针对增加了一次数据库访问的缺点，因为索引数据基本不会改变的缘故，缓存命中率非常高，所以采用memcached和redis，通过缓存减少性能损失。

缓存

1. Memcached

Memcached 是高性能分布式内存缓存服务器。其本质上就是一个内存key-value数据库，但是不支持数据的持久化，服务器关闭之后数据全部丢失。基本应用模型如下：



2. Redis

Redis是一个开源的key-value存储系统。与Memcached类似，Redis将大部分数据存储在内存中，支持的数据类型包括：字符串、哈希 表、链表、集合、有序集合以及基于这些数据类型的相关操作。基本应用模型类似Memcached.

Memcached & Redis 对比

1. Redis不仅仅支持简单的k/v类型的数据，同时还提供list，set，hash等数据结构的存储。
2. Redis支持数据的备份，即master-slave模式的数据备份。

3. Redis支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。
4、虚拟内存--Redis当物理内存用完时，可以将一些很久没用到的value 交换到磁盘
Redis在很多方面具备数据库的特征，或者说就是一个数据库系统，而Memcached只是简单的K/V缓存，具体采用哪一个要根据具体的业务应用场景决定。

数据一致性

1. CAP 理论

一致性：所有节点在同一时间具有相同的数据。
可用性：保证每个请求不管成功或者失败都有响应。即系统的性能。
分区容忍性：系统中任意信息的丢失或失败不会影响系统的继续运作。即系统的抗故障能力。
在分布式系统中，对于这三者不能同时满足。

2. 两阶段提交算法

在两阶段提交协议中，系统一般包含两类机器（或节点）：一类为协调者（coordinator），通常一个系统中只有一个；另一类为事务参与者（workers），一般包含多个，在数据存储系统中可以理解为数据副本的个数。两阶段提交协议由两个阶段组成，在正常的执行下，这两个阶段的执行过程如下所述：

- 阶段1：请求阶段（commit-request phase，或称表决阶段，voting phase）。

在请求阶段，协调者将通知事务参与者准备提交或取消事务，然后进入表决过程。在表决过程中，参与者将告知协调者自己的决策：同意（事务参与者本地作业执行成功）或取消（本地作业执行故障）。

- 阶段2：提交阶段（commit phase）。

在该阶段，协调者将基于第一个阶段的投票结果进行决策：提交或取消。当且仅当所有的参与者同意提交事务协调者才通知所有的参与者提交事务，否则协调者将通知所有的参与者取消事务。参与者在接收到协调者发来的消息后将执行响应的操作。

两阶段提交算法在分布式系统结合，可实现单用户（协调者）对文件（对象）多个副本（所有的数据副本的存储主机为参与者）的修改。

3.分布式锁服务

在整个数据处理过程中数据处于锁定状态，在用户修改数据的同时，其它用户不允许修改。采用分布式锁服务实现数据一致性，是在操作目标之前先获取操作许可，然后再执行操作，如果其他用户同时尝试操作该目标将被阻止，直到前一个用户释放许可后，其他用户才能够操作目标。Paxos算法即是常见的锁服务的一致性算法，基于消息传递且具有高度容错特性。

可扩展性 & 快速响应

以1号店应对“11.11”突增流量为例：

11.11的主要特点是流量大和突发性高，这就带来了两个核心的需求：

- 可扩展：
  如何抗住这样的流量，针对这个需求，1号店搜索团队构建了**分布式搜索引擎**，支持横向扩展；并且针对业务特点做了**Routing优化**，让搜索的效率更高。
- 快速响应
  流量越大，单位时间内的流量价值就越大，出现问题的损失也就越大，如何做到快速响应变得非常关键。针对这个需求，搜索系统支持**自动部署和快速扩容**以应对突发流量，索引数据从导入、处理到上线服务会经过层层验证，同时还有监控体系及时发现线上的问题。

分布式搜索引擎

分布式搜索是为了解决数据增长过程中索引变大和操作时间变长的问题，它将原来的单个索引文件划分成n个切片(shards)，让每个shard都足够小，从而保证索引可以在多台服务器上部署，而且搜索操作可以在较短时间内返回。

1号店分布式搜索引擎是Lucene/Solr核心的，结合SOA框架Hedwig构建了一层分布式框架，支持搜索请求的分发和合并，并且构建了搜索管理后台，支持多索引管理、集群管理、全量索引切换和实时索引更新。

高效Routing

使用分布式搜索引擎，面临的核心问题就是如何选择高效的Sharding策略和Routing方案。为了设计Routing方案，我们需要深入理解业务场景。

1号店的搜索的主入口分搜词和类目导航两种场景，对于**类目导航**，原理上非常简单，按照类目ID来查找Sharding策略，就可以确定需要访问的Shard，虽然现实中还需要考虑扩展类目等特殊场景，但是也不难做出一个简单的Routing策略。再加上类目数是有限的，Routing规则在Broker本地内存就可以缓存起来。

搜词场景下，仅凭词本身很难判断它属于哪个Shard。我们首先按照词的热度分为两类，采取不同的Routing策略。对于热词，搜词流量同样符合80-20规则，对于热词，我们可以在建完索引之后，就跑一遍热词搜索，记录那些Shard有结果，离线构建出热词Routing表。切换索引时，Routing表也一起加载进去。对于非热词，则采用首次搜索去访问所有Shard，根据结果记录Routing表，这个词在下次搜索时，就有了缓存可用。

自动部署与快速扩容

1号店采用自动部署工具。在11.11的场景下，这个自动化部署工具支持快速扩容，基本过程为：

（1） Index Server（部署工具界面）计算出扩容之后的部署方案，写入到ZooKeeper。这里的算法与初始部署有些不同，它需要保证现有服务器的Shard尽量不变。

（2） 每个Shard Searcher服务器都会监听ZooKeeper上的部署方案，方案改变之后，Shard Searcher会获取新的方案与本地方案对比，进行增减操作。

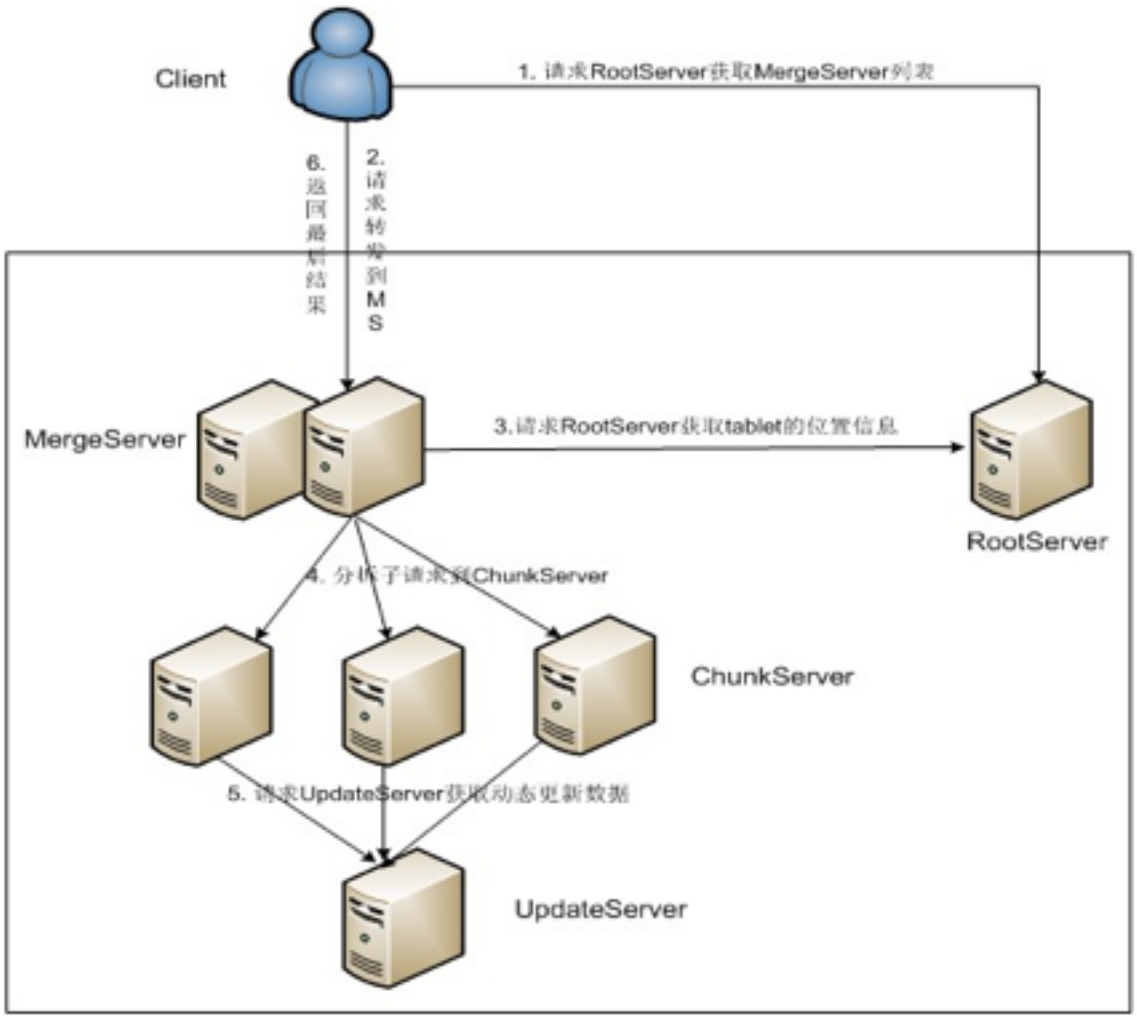
（3） Shard Searcher从HDFS上获取索引数据，与最近的实时更新数据合并之后，启动索引提供服务。

有了自动扩容工具，11.11容量规划确定之后，新的服务器就可以很快部署上线，也减少了人工操作可能引入的失误等问题。大促期间如果需要紧急扩容，也可以在几分钟内提高整个系统的服务能力。

负载均衡

在分布式系统中存在着著名的“短板理论”，一个集群如果出现了负载不均衡问题，那么负载最大的机器往往将成为影响系统整体表现的瓶颈和短板。为了避免这种情况的发生，需要动态负载均衡机制，以达到实时的最大化资源利用率，从而提升系统整体的吞吐。

以淘宝 OceanBase分布式系统在双十一前期准备工作为例来分析负载平衡。OceanBase是一个具有自治功能的分布式存储系统，由中心节点RootServer、静态数据节点ChunkServer、动态数据节点UpdateServer以及数据合并节点MergeServer四个Server构成。客户端连接到OceanBase之后一次读请求的读流程如下图所示：



1. Client 从RootServer获取到MergeServer 列表；
2. Client将请求发送到某一台MergeServer；

3. MergeServer从RootServer获取请求对应的ChunkServer位置信息；
4. MergeServer将请求按照Tablet拆分成多个子请求发送到对应的ChunkServer；
5. ChunkServer向UpdateServer请求最新的动态数据，与静态数据进行合并；
6. MergeServer合并所有子请求的数据，返回给Client；

第1步和第3步中Client与RootServer以及MergeServer与RootServer的两次交互会利用缓存机制来避免，即提高了效率，同时也极大降低了RootServer的负载。在第2步客户端选择MergeServer时如果调度不均衡会导致某台MergeServer机器过载；在第4步MergeServer把子请求发送到数据所在的ChunkServer时，由于每个tablet会有多个副本，选择副本的策略如果不均衡也会造成ChunkServer机器过载。通过查看OceanBase内部各模块的提供的监控信息，发现**负载不均问题是由MergeServer对ChunkServer的访问出现了不均衡导致了部分ChunkServer的过载**。

ChunkServer是存储静态Tablet分片数据的节点，分析其负载不均的原因包含如下可能：

- 数据不均衡： ChunkServer上数据大小的分布是不均衡的，比如某些节点因为存储Tablet分片数据量多少的差异性而造成的不均衡；
- 流量不均衡：数据即使是基本均衡的情况下，仍然会因为某些节点存在数据热点等原因而造成流量是不均衡的。

通过对RootServer管理的所有tablet数据分片所在位置信息Metadata进行统计，发现各个ChunkServer上的tablet数据量差异不大，问题锁定为**流量不均衡导致ChunkServer的过载**。

如果把热点ChunkServer上非热点Tablet的访问调度到其他Server，是可以缓解流量不均问题的，因此我们设计了**新的负载均衡算法为：以实时统计的ChunkServer上所有tablet的访问次数为Ticket，每次对Tablet的读请求会选择副本中得票率最低的ChunkServer**。

考虑到流量不均衡的第二个原因是请求的差异较大问题，ChunkServer对外提供的接口分为Get和Scan两种，Scan是扫描一个范围的所有行数据，Get是获取指定一行数据，因此两种访问方式的次数需要划分赋予不同的权重(α,β)，平均响应时间（avg_time）参与最终Ticket的运算，同时尽量减少对实时性的影响：

$$\sum_{i=0}^{T} \frac{(\alpha * get\_count_i + \beta * scan\_count_i) * (avg\_time_i / 1000)}{\gamma^{(T-i)}}$$

采用新的算法后，很好的缓解了负载不均衡的问题，整体负载提升了1倍，整体QPS吞吐提升到了8w。

参考资料

[1] 分布式系统的特点以及设计理念 http://www.infoq.com/cn/articles/features-and-design-concept-of-distributed-system

[2] 分布式存储系统 知识体系 http://wuchong.me/blog/2014/08/07/distributed-storage-system-knowledge/

［3］又拍网架构中的分库设计 http://www.infoq.com/cn/articles/yupoo-partition-database
［4］分布式mysql的美丽说实现 http://wenku.baidu.com/view/7f85b41d55270722192ef78e.html
［5］mysql 与缓存服务器集成的介绍（memcache+redis）http://blog.csdn.net/cymm_liu/article/details/45744259
［6］分布式系统的一致性算法简介 http://welcome66.iteye.com/blog/2216571
［7］Paxos算法 https://zh.wikipedia.org/zh/Paxos%E7%AE%97%E6%B3%95
［8］1号店11.11：分布式搜索引擎的架构实践 http://www.infoq.com/cn/articles/yhd-11-11-distributed-search-engine-architecture
［9］淘宝：OceanBase分布式系统负载均衡案例分享 http://www.csdn.net/article/2013-02-28/2814303-sharing-OceanBase-distributed-system-load-balance-case