讨论课 2

------分布式系统

软件工程　潘舜达　1354366

一．分布式系统的意义：

1、升级单机处理能力的性价比

根据摩尔定律来说，如果把时间固定下来，所需要的处理器性能越高，付出的成本就越高，性价比就越低。而且单机处理器始终有瓶颈

2.单机处理能力存在瓶颈

单机处理器的瓶颈只能通过多机来解决

3.稳定性和可用性

如果采用单机系统，如果这台机器一切正常，则一切 ok,如果这台机器坏了。整个应用就访问不了了。如果要做容灾备份等方案，就需要考虑分布式系统了。

二．分布式系统设计理念

1. 分布式系统对服务器硬件要求很低

这一点主要现在如下两个方面：

（1）对服务器硬件可靠性不做要求，允许服务器硬件发生故障，硬件的故障由软件来容错。所以分布式系统的高可靠性是由软件来保证。

(2)对服务器的性能不做要求，不要求使用高频 CPU、大容量内存、高性能存储等等。因为分布式系统的性能瓶颈在于节点间通讯带来的网络开销，单台服务器硬件性能再好，也要等待网络 IO。

2.分布式系统强调横向可扩展性

横向可扩展性（Scale Out）是指通过增加服务器数量来提升集群整体性能。纵向可扩展性（Scale Up）是指提升每台服务器性能进而提升集群整体性能。纵向可扩展性的上限非常明显，单台服务器的性能不可能无限提升，而且跟服务器性能相比，网络开销才是分布式系统最大的瓶颈。横向可扩展性的上限空间比较大，集群总能很方便地增加服务器。而且分布式系统会尽可能保证横向扩展带来集群整体性能的（准）线性提升。比如有 10 台服务器组成的集群，横向扩展为 100 台同样服务器的集群，那么整体分布式系统性能会提升为接近原来的 10 倍。

3. 分布式系统不允许单点失效（No Single Point Failure）

单点失效是指，某个应用服务只有一份实例运行在某一台服务器上，这台服务器一旦挂掉，那么这个应用服务必然也受影响而挂掉，导致整个服务不可用。例如，某网站后台如果只在某一台服务器上运行一份，那这台服务器一旦宕机，该网站服务必然受影响而不可用。再比如，如果所有数据都存在某一台服务器上，那一旦这台服务器坏了，所有数据都不可访问。

4. 分布式系统尽可能减少节点间通讯开销

如前所述，分布式系统的整体性能瓶颈在于内部网络开销。目前网络传输的速度还赶不上CPU读取内存或硬盘的速度，所以减少网络通讯开销，让CPU尽可能处理内存的数据或本地硬盘的数据，能显著提高分布式系统的性能。典型的例子就是Hadoop MapReduce，把计算任务分配到要处理的数据所在的节点上运行，从而避免在网络上传输数据。

5. 分布式系统应用服务最好做成无状态的

应用服务的状态是指运行时程序因为处理服务请求而存在内存的数据。分布式应用服务最好是设计成无状态。因为如果应用程序是有状态的，那么一旦服务器宕机就会使得应用服务程序受影响而挂掉，那存在内存的数据也就丢失了，这显然不是高可靠的服务。把应用服务设计成无状态的，让程序把需要保存的数据都保存在专门的存储上，这样应用服务程序可以任意重启而不丢失数据，方便分布式系统在服务器宕机后恢复应用服务。

三．数据一致性

1.CAP原则

CAP原则是NOSQL数据库的基石。Consistency(一致性)。Availability(可用性)。Partition tolerance（分区容错性）

分布式系统的CAP理论：理论首先把分布式系统中的三个特性进行了如下归纳：

● 一致性（C）：在分布式系统中的所有数据备份，在同一时刻是否同样的值。（等同于所有节点访问同一份最新的数据副本）

● 可用性（A）：在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求。（对数据更新具备高可用性）

● 分区容错性（P）：以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在C和A之间做出选择。

2.两阶段提交协议/算法（2PC）

2PC顾名思义分为两个阶段，其实施思路可概括为：

（1）投票阶段（voting phase）：参与者将操作结果通知协调者；

（2）提交阶段（commit phase）：收到参与者的通知后，协调者再向参与者发出通知，根据反馈情况决定各参与者是否要提交还是回滚；

3.消息队列法

消息队列是进程间通信的一种机制，两个或多个进程间通过访问共用的系统消息队列来交换信息，这里将消息队列的概念扩展到位于分布式环境下的不同站点间的进程间通信[5]，由消息管理机构在Internet 上实现可靠的消息传送，其中涉及的消息队列有发送队列、接收队列、应答队列和管理队列。

应用程序在本地数据库上完成数据更新后，将更新信息以及本地应答队列和管理队列的地址存放到消息中，发送到远端接收队列中，消息交给消息管理机构后首先进入本地发送队列等待发送，如一切正常，消息将送到应用程序指定的远端接收队列中，同时，一个后台处理程序一直在监视着自己的接收队列，一旦有消息到达，它将读消息，并对本地数据库实施消息中所描述的更新操作，如果更新成功，则处理结束，否则依应答队列的地址信息

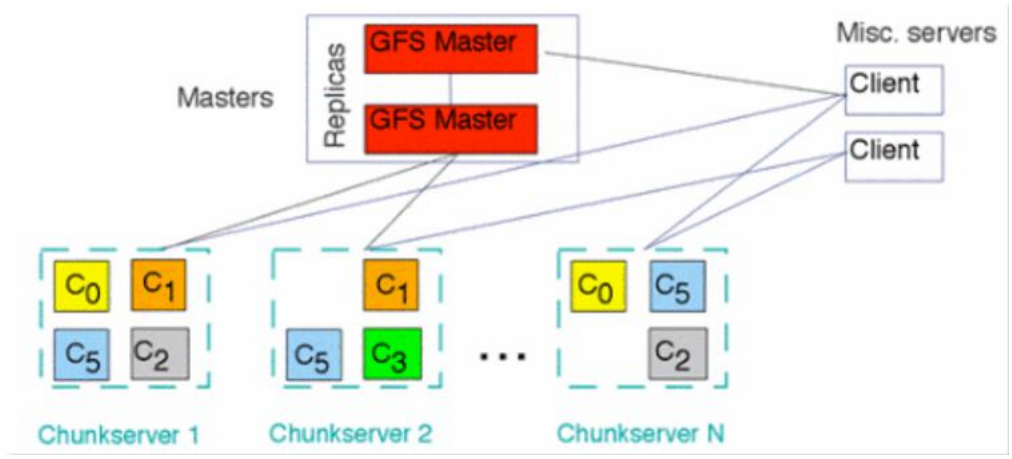
发送出错消息，另一个后台处理程序一直在监视应答队列，根据收到的错误信息，它将在本地数据库中试图撤消相应的更新操作。

该方法提供的是数据库间异步更新，而不是同步更新，这样可能读到的不是最新数据，所以它不适合要求实时数据同步的分布式数据库系统。

四．目前业界存在的几种分布式系统

Company using	Distributed Filesystem	Master Node (Y/N)
Google	GFS&Bigtable	Y
Amazon	Dynamo	N
Microsoft	Azure	Y
Yahoo	PNUTS	Y

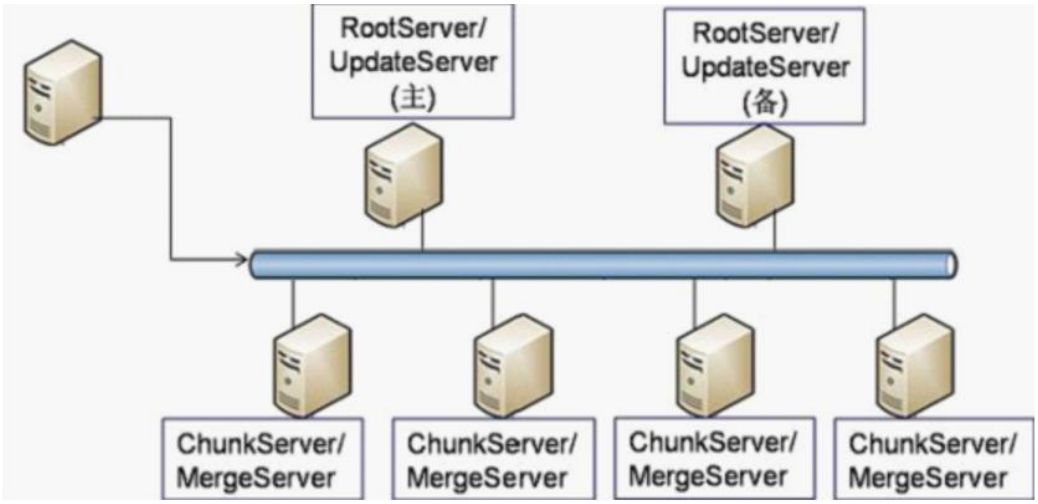
有中心节点的分布式架构：



五． 业界实例------淘宝的 OceanBase 分布式系统

（1）OceanBase 架构介绍

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[1]，如下图所示。



Tablet：分片数据，最基本的存储单元，一般会存储多份，一个 Table 由多个 tablet 构成；

RootServer：负责集群机器的管理、Tablet 定位、数据负载均衡、Schema 等元数据管理等。

UpdateServer：负责存储动态更新数据，存储介质为内存和 SSD，对外提供写服务；

ChunkServer：负责存储静态 Tablet 数据，存储介质为普通磁盘或者 SSD。

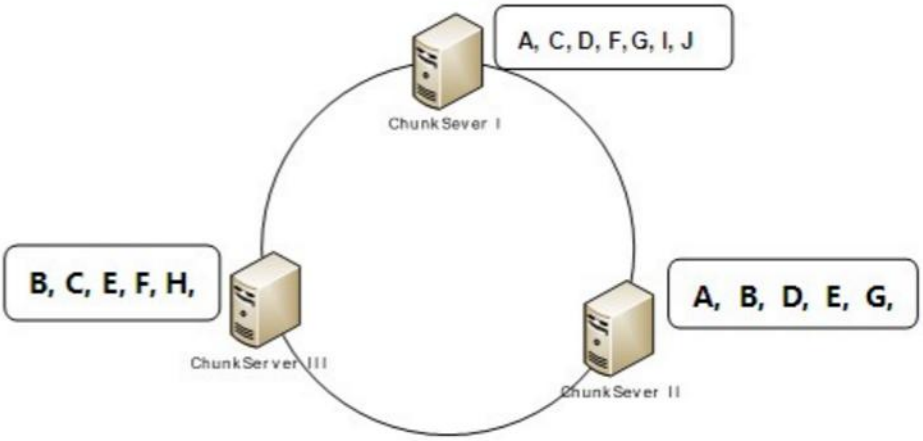
MergeServer：负责对查询中涉及多个 Tablet 数据进行合并，对外提供读服务；

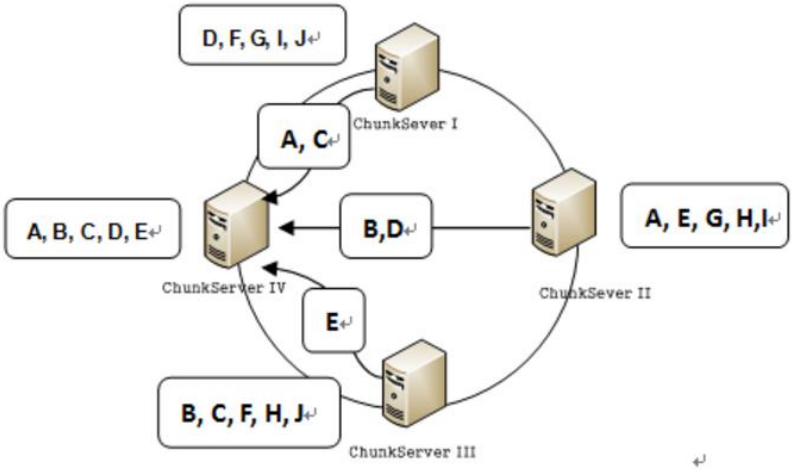
（2）双十一前期准备

对于淘宝的大部分应用而言，“双十一”就是一年一度的一次线上压测。伴随流量不断刷新着历史新高，对每个系统的可扩展性提出了很大的挑战。为了迎战双十一各产品线对有可能成为瓶颈部分的流量进行预估和扩容成为刻不容缓的任务。在本文要分享的案例中，应用方根据历史数据预估读请求的访问峰值为 7w QPS，约为平时的 5-6 倍，合计每天支持 56 亿次的读请求。当时 OceanBase 集群部署规模是 36 台服务器，存储总数据量为 200 亿行记录，每天支持 24 亿次的读请求。

当前集群的读取性能远不能满足需求，我们首先进行了一次扩容，上线了 10 台 Chunkserver/Mergeserver 服务器。由于 OceanBase 本身具有比较强的可扩展性，为集群加机器是一件非常简单的操作。中心节点 Rootserver 在新机器注册上线后，会启动 Rebalance 功能以 Tablet 为单位对静态数据进行数据迁移，见下图的示意，最终达到所有 ChunkServer 上数据分片的均衡分布。

Tablet	Start_key	End_key
A	MIN	100
B	101	200
C	201	300
D	301	400
E	401	500
F	501	600
G	601	700
H	701	800
I	801	900
J	901	MAX



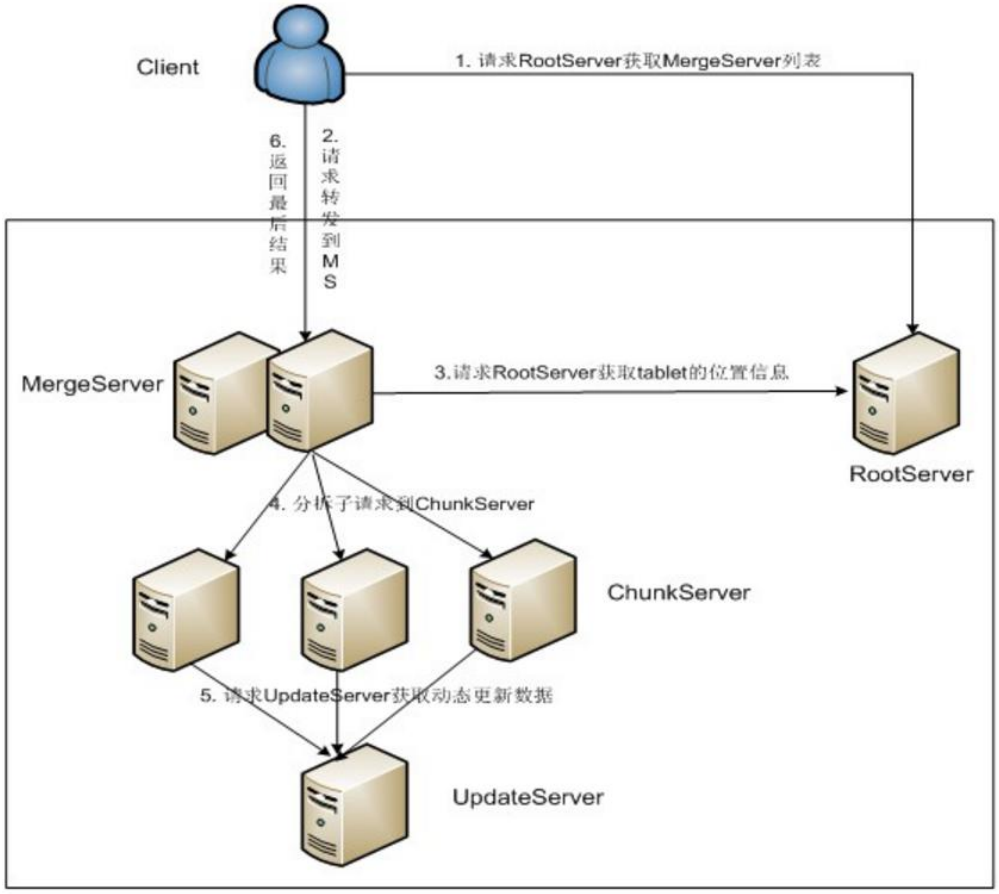


扩容完成后引入线上流量回放机制进行压力测试，以验证当前集群的性能是否可以满足应用的双十一需求。我们使用了 10 台服务器，共 2000-4000 个线程并发回放线上读流量对集群

进行压测，很快发现集群整体的 QPS 在达到 4 万左右后，压测客户端出现大量超时现象，平均响应延迟已经超过阈值 100ms，即使不断调整压力，系统的整体 QPS 也没有任何增大。此时观察整个集群机器的负载状态发现只有极个别服务器的负载超高，是其他机器的 4 倍左右，其他机器基本处于空闲状态，CPU、网络、磁盘 IO 都凸现了严重的不均衡问题。

（3）负载不均问题跟踪

客户端连接到 OceanBase 之后一次读请求的读流程如下图所示：



Client 从 RootServer 获取到 MergeServer 列表；

Client 将请求发送到某一台 MergeServer；

MergeServer 从 RootServer 获取请求对应的 ChunkServer 位置信息；

MergeServer 将请求按照 Tablet 拆分成多个子请求发送到对应的 ChunkServer；

ChunkServer 向 UpdateServer 请求最新的动态数据，与静态数据进行合并；

MergeServer 合并所有子请求的数据，返回给 Client；

可见热点 Tablet 虽访问频率稍高对负载的贡献率相对较大，但是热点 tablet 的占比很低，相反所有非热点 tablet 对负载的贡献率总和还是很高的，这种情况就好比“长尾效应”

（4）负载均衡算法设计

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。

同时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)参与最终 Ticket 的运算：

$$\alpha * \text{get\_count} + \beta * \text{scan\_count}$$

除此之外，简单的区分两种访问模式还是远远不够的，不同的 Scan 占用的资源也是存在较大差异的，引入平均响应时间（avg_time）这个重要因素也是十分必要的：

$$(\alpha * \text{get\_count} + \beta * \text{scan\_count}) * (\text{avg\_time}/1000)$$

负载均衡算法要求具有自适应性和强的实时性，一方面新的访问要实时累积参与下次的负载均衡的调度，另一方面历史权重数据则需要根据统计周期进行非线性的衰减（y 衰减因子），减少对实时性的影响：

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

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。

（5）小结

负载均衡问题是老生常谈的问题，解决不好就会出现“短板效应”，更甚至引发分布式系统中的连锁反应即“雪崩”，从而演化成系统的灾难。负载均衡的算法也层出不穷，有的出于成本最优，有的是为了最小延迟，有的则是最大化系统吞吐，目的不同算法自然各异，不存在包治百病的良方，并不是越复杂的算法越有效[4]，要综合考虑算法所需数据获取的 Overhead，更多的是遵循“简单实用”的法则，根据业务场景进行分析和尝试。

六. 参考资料

分布式系统设计原理与方案
http://www.cnblogs.com/pains/archive/2011/06/06/2073482.html

分布式系统浅析 http://blog.csdn.net/kinges/article/details/6097899

保持数据一致性的方法
http://blog.163.com/faye422@126/blog/static/355189922012324102329418/

两阶段提交(2PC)协议 http://blog.chinaunix.net/uid-20761674-id-75164.html

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

MapReduce/GFS/BigTable 三大技术资料
http://blog.csdn.net/wangxiaoqin00007/article/details/7091686