第二章 关键技术介绍

2.1 Thrift — RPC Framework

Apache Thrift 是 Facebook 实现的一种高效的、支持多种编程语言的远程服务调用的框架。

目前流行的服务调用方式有很多种，例如基于 SOAP 消息格式的 Web Service，基于 JSON 消息格式的 RESTful 服务等。其中所用到的数据传输方式包括 XML，JSON 等，然而 XML 相对体积太大，传输效率低，JSON 体积较小，新颖，但还不够完善。本文将介绍由 Facebook 开发的远程服务调用框架 Apache Thrift，它采用接口描述语言定义并创建服务，支持可扩展的跨语言服务开发，所包含的代码生成引擎可以在多种语言中，如 C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, Smalltalk 等创建高效的、无缝的服务，其传输数据采用二进制格式，相对 XML 和 JSON 体积更小，对于高并发、大数据量和多语言的环境更有优势。

2.2 分布式服务框架 Zookeeper

Zookeeper 分布式服务框架是 Apache Hadoop 的一个子项目，它主要是用来解决分布式应用中经常遇到的一些数据管理问题，如：统一命名服务、状态同步服务、集群管理、分布式应用配置项的管理等。

2.3 服务器缓存服务 Redis

Redis 是一个开源（BSD许可）的，内存中的数据结构存储系统，它可以用作数据库、缓存和消息中间件. 它支持多种类型的数据结构，如字符串（strings），散列（hashes），列表（lists），集合（sets），有序集合（sorted sets）与范围查询，bitmaps，hyperloglogs和地理空间（geospatial）索引半径查询. Redis 内置了复制（replication），LUA脚本（Lua scripting），LRU驱动事件（LRU eviction），事务（transactions）和不同级别的磁盘持久化（persistence）， 并通过Redis哨兵（Sentinel）和自动分区（Cluster）提供高可用性（high availability）.

你可以对这些类型执行原子操作， 列如：字符串（strings）的append 命令;散列（hashes）的hincrby命令;列表（lists）的lpush命令;集合（sets）计算交集sinter命令，计算并集union命令和计算差集sdiff命令; 或者在有序集合（sorted sets）里面获取成员的最高排名zrangebyscore命令.

为了实现其卓越的性能， Redis 采用运行在内存中的数据集工作方式. 根据您的使用情况， 您可以每隔一定时间将数据集导出到磁盘， 或者追加到命令日志中. 您也可以关闭持久化功能，将Redis作为一个高效的网络的缓存数据功能使用.Redis 同样支持主从复制（能自动重连和网络断开时自动重新同步），并且第一次同步是快速的非阻塞试的同步.

2.4 NoSql数据库 MongoDB

Mongodb无模式优点

1、存储形式，相较于传统的关系型数据库，它可以使用一个表存储原来需要使用多表关联才能存储的数据库。

2、存储动态性，相较于传统的数据库当要增加一个属性值的时候要对表大动，mongodb的面向文档的形式可以使其属性值轻意的增加和删除。而原来 的关系型数据库要实现这个需要有很多的属性表来支持。

Mongodb存储特点

3、Mongodb有两种数据写入模式，

• 一种是高效的fire-and-forget模式就是只管向数据库服务器提交数据不等数据库服务器的回应。这个是数据库默认模式。

• 另外一种是安全模式，就是写入的同时还要与服务器同步，当数据的安全性要求高的时候适合用安全模式。

4、Mongodb的日志功能。

• 日志功能是帮助在系统Down机的时候恢复数据完整性做的，默认情况下Mongodb是开启日志功能的。每一个数据库操作都会先记录日志，所以当down机重启动服务器的时候数据库服务器能够通过日志文件恢复之前未完成的操作。

• 当然，开启日志必然影响性能，所以用户需要根据自己的实际应用来选择一个合适的工作模式。

• 如果你打算使用无日志模式，最好使用拷贝集群模式，这样可以降低数据丢失的风险。

Mongodb的索引

5、Mongodb支持关系型数据库的所有索引模式（升序，降序，唯一，复合，地理空间索引），同时也支持二级索引（通过B-tree实现）。每一个collection支持64个索引

6、支持ad hoc queries用户自己可以定义自己的查询。

Mongodb的集群

7、Mongodb的集群模式是主从模式的集群，其中主服务器只有一个，支持读写，而从服务器可以有多个只支持读取操作。集群的主要目的是通过数据的冗余实现failover模式，从而提高数据库服务器的可用性。当主服务器断掉后其中的一个从服务器会自动升级为主服务器，其它drive client只要通过failover就可以自动切换到新的主服务器，正常工作。而当原来的主服务器再恢复工作的时候，它将自动转为从服务器。

Mongodb的可扩展性

8、Mongodb采用了最新的低成本的横向扩展模式，相对于传统的单结点纵向扩展，可以节约成功，而且有更好的可靠性，更好的数据处理性能。

更好的数据处理性能。

第三章 分布式系统介绍

3.1 分布式系统的定义

我们把分布式系统定义成一个其硬件或软件组件分布在连网的计算机上，组件之间通过传递消息进行通信和动作协调的系统。这个简单的定义覆盖了所有可有效部署连网计算机的系统。

由一个网络连接的计算机可能在空间上的距离不等。它们可能分布在地球上不同的洲，也可能在同一栋楼或同一个房间里。我们定义的分布式系统有如下显著特征：

并发：在一个计算机网络中，执行并发程序是常见的行为。用户可以在各自的计算机上工作，在必要时共享诸如Web页面或文件之类的资源。系统处理共享资源的能力会随着网络资源（例如，计算机）的增加而提高。在本书的许多地方将描述有效部署这种额外能力的方法。

缺乏全局时钟：在程序需要协作时，它们通过交换消息来协调它们的动作。密切的协作通常取决于对程序动作发生的时间的共识。但是，事实证明，网络上的计算机与时钟同步所达到的准确性是有限的，即没有一个正确时间的全局概念。这是通信仅仅是通过网络发送消息这个事实带来的直接结果。

故障独立性：所有的计算机系统都可能出故障，一般由系统设计者负责为可能的故障设计结果。分布式系统可能以新的方式出现故障。网络故障导致网上互连的计算机的隔离，但这并不意味着它们停止运行，事实上，计算机上的程序不能够检测到网络是出现故障还是网络运行得比通常慢。类似的，计算机的故障或系统中程序的异常终止（崩溃），并不能马上使与它通信的其他组件了解。系统的每个组件会单独地出现故障，而其他组件还在运行。

我们看一下现代分布式系统的几个例子，包括Web搜索、多人在线游戏和金融交易系统，也考察今天推动分布式系统发展的关键趋势：现代网络的泛在特性，移动和无处不在计算的出现，分布式多媒体系统不断增加的重要性，以及把分布式系统看成一种实用系统的趋势。接着本章强调资源共享是构造分布式系统的主要动机。资源可以被服务器管理，由客户访问，或者它们被封装成对象，由其他客户对象访问。

构造分布式系统的挑战是处理其组件的异构性、开放性（允许增加或替换组件）、安全性、可伸缩性（用户的负载或数量增加时能正常运行的能力）、故障处理、组件的并发性、透明性和提供服务质量的问题。

3.2 CAP定律

在理论计算机科学中，CAP定理（CAP theorem），又被称作布鲁尔定理（Brewer's theorem），它指出对于一个分布式计算系统来说，不可能同时满足以下三点：[1][2]

• 一致性（Consistence) (等同于所有节点访问同一份最新的数据副本）

• 可用性（Availability）（对数据更新具备高可用性）

• 容忍网络分区（Partition tolerance）（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[3]。）

根据定理，分布式系统只能满足三项中的两项而不可能满足全部三项[4]。理解CAP理论的最简单方式是想象两个节点分处分区两侧。允许至少一个节点更新状态会导致数据不一致，即丧失了C性质。如果为了保证数据一致性，将分区一侧的节点设置为不可用，那么又丧失了A性质。除非两个节点可以互相通信，才能既保证C又保证A，这又会导致丧失P性质。

3.3 现代分布式系统的特点

1. 对服务器硬件要求低

• 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。

• 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. 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. 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。

因为分布式系统的服务器都是廉价的PC服务器，硬件不能保证100%可靠，所以分布式系统默认每台服务器随时都可能发生故障挂掉。同时分布式系统必须要提供高可靠服务，不允许出现单点失效，因此分布式系统里运行的每个应用服务都有多个运行实例跑在多个节点上，每个数据点都有多个备份存在不同的节点上。这样一来，多个节点同时发生故障，导致某个应用服务的所有实例都挂掉、或某个数据点的多个备份都不可读的概率大大降低，进而有效防止单点失效。

通常情况，不要让服务器满负荷运行，服务器长时间满负荷运行的话，出故障的概率显著升高。所以分布式系统采用一大堆中低性能的PC服务器，尽可能把负载均摊到所有服务器上，让每台服务器的负载都不高，保证集群整体稳定性。

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。

比如，在设计网站后台的时候，对于用户登陆请求，可以把登陆用户的session相关信息保存在Redis或Memcache等缓存服务中，这样每个网站的后台实例不保存用户登录状态，这样即使重启网站后台程序也不丢失用户的登录状态信息；如果把用户的session相关信息保存在网站后台程序的内存里，那一旦受理用户登录的网站后台程序实例挂掉，必然有用户的登录状态信息会丢失。

总而言之，分布式系统是大数据时代企业级应用的首选平台，它有良好的可扩展性，尤其是横向可扩展性（Scale Out），使得分布式系统非常灵活，能应对千变万化的企业级需求，而且降低了企业客户对服务器硬件的要求，真正能做到应用服务层面的弹性扩展（auto-scaling）。