复用产品方案

刘旭东

分布式(Distributed System)

> 分布式系统是一个硬件或软件组件分布在不同的网络计算机上，彼此之间仅仅通过消息传递进行通信和协调的系统。

通俗来讲，就相当于一群独立计算机集合共同对外提供服务，但是对于系统的用户来说，就像是一台计算机在提供服务一样。分布式意味着可以采用更多的普通计算机（相对于昂贵的大型机）组成分布式集群对外提供服务。计算机越多，CPU、内存、存储资源等也就越多，能够处理的并发访问量也就越大。因为各个主机之间通信和协调主要通过网络进行，所以分布式系统中的计算机在空间上几乎没有任何限制，这些计算机可能被放在不同的机柜上，也可能被部署在不同的机房中，还可能在不同的城市中，对于大型的网站甚至可能分布在不同的国家和地区。

典型的分布式系统有如下的特点

- 可扩展性

  能够适应需求变化而扩展。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。

- 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。

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

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。

- 分布式系统尽可能减少节点间通讯开销

  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。

分布式系统的局限和缺点

CAP定理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：

- 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。）

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。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。

如果既要保持高可用性又要求保持高吞吐量，即，要求这个分布式系统对A和P这两个特性尽量的满足，那么根据CAP定理，这个分布式系统必然对**一致性(C)**很难有非常良好的支持。

业内实践

- Apache Hadoop

  Apache Hadoop是一款支持数据密集型分布式应用并以Apache 2.0许可协议发布的开源软件框架。它支持在商品硬件构建的大型集群上运行的应用

程序。Hadoop是根据Google公司发表的MapReduce和Google文件系统的论文自行实现而成。

- Apache Spark

  Apache Spark是一个开源簇运算框架，最初是由加州大学柏克莱分校AMPLab所开发。相对于Hadoop的MapReduce会在运行完工作后将中介数据存放到磁盘中，Spark使用了内存内运算技术，能在数据尚未写入硬盘时即在内存内分析运算。Spark在内存内运行程序的运算速度能做到比Hadoop MapReduce的运算速度快上100倍，即便是运行程序于硬盘时，Spark也能快上10倍速度。Spark允许用户将数据加载至簇内存，并多次对其进行查询，非常适合用于机器学习算法。

负载均衡 (Load Balancing)

> 负载平衡（Load balancing）是一种计算机网络技术，用来在多个计算机（计算机集群）、网络连接、CPU、磁盘驱动器或其他资源中分配负载，以达到最佳化资源使用、最大化吞吐率、最小化响应时间、同时避免过载的目的。

负载平衡最重要的一个应用是利用多台服务器提供单一服务，这种方案有时也称之为服务器农场。对于互联网服务，负载平衡器通常是一个软件程序，这个程序侦听一个外部端口，互联网用户可以通过这个端口来访问服务，而作为负载平衡器的软体会将用户的请求转发给后台内网服务器，内网服务器将请求的响应返回给负载平衡器，负载平衡器再将响应发送到用户，这样就向互联网用户隐藏了内网结构，阻止了用户直接访问后台（内网）服务器，使得服务器更加安全，可以阻止对核心网络栈和运行在其它端口服务的攻击。

当所有后台服务器出现故障时，有些负载平衡器会提供一些特殊的功能来处理这种情况。例如转发请求到一个备用的负载平衡器、显示一条关于服务中断的消息等。负载平衡器使得IT团队可以显著提高容错能力。它可以自动提供大量的容量以处理任何应用程序流量的增加或减少。

现在负载平衡器也开始支持数据库服务，称之为数据库负载平衡器。

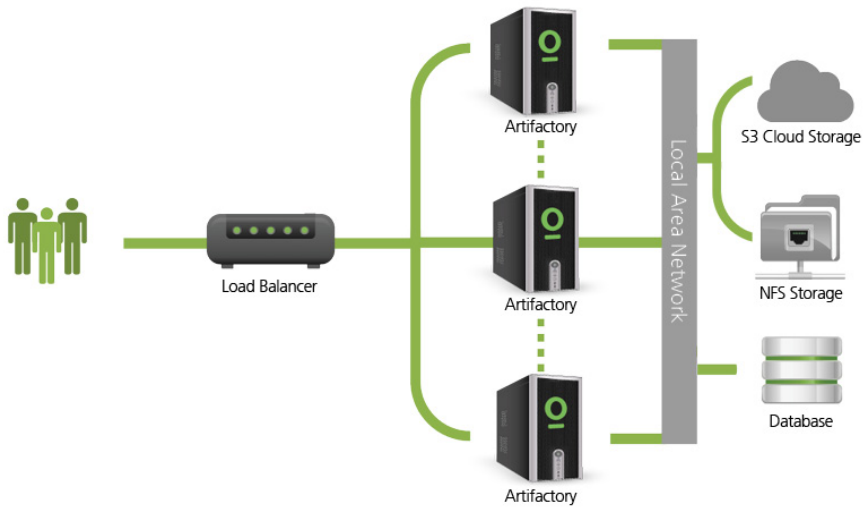
负载均衡在电信行业也有着广泛的用途:

- 对通讯链路的冗余是非常有用的,使用负载均衡器，两条（多条）连接可以都投入使用。有一个设备或者程序实时监控着所有连接的连通性，并且对

正在发送的包进行选路。同时使用多条连接可以增加带宽。

- 许多电信公司在其内部网络或连接到外部网络（其它电信网络）都有多条线路可以使用。为避免某条链路出现网络堵塞，最小化连接其它网络的费用或者提高网络的可靠性，它们使用负载均衡将流量从一条链路转移到另一条链路。
- 负载均衡的另一个用途是监控网络活动。负载均衡器能用于将巨大的网络流量分割为几个子流并使用网络分析器，每个都读取原始数据的一部分。这对于监视10GbE, STM64高速网络非常有用，在这些网络上由于数据量太大进行复杂的数据处理几乎是不可能的。

负载均衡的架构



负载均衡的问题**持续性**:如果会话信息保存在后台服务器，用户接下来的请求可能会被分配到不同的后台服务器，此时用户会话就无法继续。

流行的解决方案：通常客户浏览器可以保存用户的每个会话信息。例如使用浏览器cookie，对数据加密并加上一个时间戳就可以保证安全了,将会话信息存储在客户端通常是首选方案，因为这样负载均衡器就可以灵活的选择后台服务器来处理用户请求。

负载均衡调度算法

当前，负载均衡器有各种各样的工作排程算法（用于决定将前端用户请求发送到哪一个后台服务器），最简单的是随机选择和轮询。更为高级的负载均衡器

会考虑其它更多的相关因素，如后台服务器的负载，响应时间，运行状态，活动连接数，地理位置，处理能力，或最近分配的流量。

静态算法

- 轮询（Round Robin）：顺序循环将请求一次顺序循环地连接每个服务器。当其中某个服务器发生第二到第7 层的故障，BIG-IP 就把其从顺序循环队列中拿出，不参加下一次的轮询，直到其恢复正常。
- 比率（Ratio）：给每个服务器分配一个加权值为比例，根据这个比例，把用户的请求分配到每个服务器。当其中某个服务器发生第二到第7 层的故障，BIG-IP 就把其从服务器队列中拿出，不参加下一次的用户请求的分配，直到其恢复正常。
- 优先权（Priority）：给所有服务器分组,给每个组定义优先权，BIG-IP 用户的请求，分配给优先级最高的服务器组（在同一组内，采用轮询或比率算法，分配用户的请求）；当最高优先级中所有服务器出现故障，BIG-IP 才将请求送给次优先级的服务器组。这种方式，实际为用户提供一种热备份的方式。

动态负载均衡算法

- 最少的连接方式（Least Connection）：传递新的连接给那些进行最少连接处理的服务器。当其中某个服务器发生第二到第7 层的故障，BIG-IP 就把其从服务器队列中拿出，不参加下一次的用户请求的分配，直到其恢复正常。
- 最快模式（Fastest）：传递连接给那些响应最快的服务器。当其中某个服务器发生第二到第7 层的故障，BIG-IP 就把其从服务器队列中拿出，不参加下一次的用户请求的分配，直到其恢复正常。
- 观察模式（Observed）：连接数目和响应时间以这两项的最佳平衡为依据为新的请求选择服务器。当其中某个服务器发生第二到第7 层的故障，BIG-IP就把其从服务器队列中拿出，不参加下一次的用户请求的分配，直到其恢复正常。
- 预测模式（Predictive）：BIG-IP利用收集到的服务器当前的性能指标，进行预测分析，选择一台服务器在下一个时间片内，其性能将达到最佳的服务器相应用户的请求。(被BIG-IP 进行检测)
- 动态性能分配(Dynamic Ratio-APM):BIG-IP 收集到的应用程序和应用服务器的各项性能参数，动态调整流量分配。

业内实践

- Apache Web服务器的mod proxy balancer扩展
- nginx
- Varnish

参考 Reference

- 初识分布式系统.
- 分布式系统的特点以及设计理念 --王璞
- Hadoop 维基百科
- Spark 维基百科
- CAP理论十二年回顾："规则"变了 --Eric Brewer(CAP理论提出者)
- 负载均衡 维基百科)
- Artifactory High Availability -- Rami Honig
- 几种负载均衡算法