　　虽然本人在前面也写过好几篇分布式系统相关的文章，主要包括[CAP理论](http://www.cnblogs.com/xybaby/p/6871764.html)、[分布式存储](http://www.cnblogs.com/xybaby/p/6930977.html)与[分布式事务](http://www.cnblogs.com/xybaby/p/7465816.html)，但对于分布式系统，并没有一个跟清晰的概念。分布式系统涉及到很多的技术、理论与协议，很多人也说，分布式系统是“入门容易，深入难”，我之前的学习也只算是管中窥豹，只见得其中一斑。因此，一致希望能对分布式系统有一个更全面的认识，至少能够把分布式系统中的各个技术、理论串起来，了解他们在分布式系统分别解决什么问题，有哪些优秀的实现。

　　我曾在网络上搜索过”如何学习分布式系统“，也在知乎上关注了该话题，但并没有看到一个全面的、有指导意义的答案。本文的目标是给打算全面学习分布式系统的自己、以及感兴趣的读者指明一条可行的路径，使得之后的学习不再盲目。

　　不过，我并没有越过这座山，我只是站在山前，从前人留下的痕迹揣测山的全貌与沟壑，臆想的成分居多，还望各位大师指点迷津。

　　2018 03 14更新：对于如何学习分布式系统，经过思考，我觉得有更好的方法，请参见《[分布式学习最佳实践：从分布式系统的特征开始（附思维导图）](https://www.cnblogs.com/xybaby/p/8544715.html)》

　　本文地址：<http://www.cnblogs.com/xybaby/p/7787034.html>

**什么是分布式系统**

　　分布式系统是由一组通过网络进行通信、为了完成共同的任务而协调工作的计算机节点组成的系统。分布式系统的出现是为了用廉价的、普通的机器完成单个计算机无法完成的计算、存储任务。其目的是**利用更多的机器，处理更多的数据**。

　　首先需要明确的是，只有当单个节点的处理能力无法满足日益增长的计算、存储任务的时候，且硬件的提升（加内存、加磁盘、使用更好的CPU）高昂到得不偿失的时候，应用程序也不能进一步优化的时候，我们才需要考虑分布式系统。因为，分布式系统要解决的问题本身就是和单机系统一样的，而由于分布式系统多节点、通过网络通信的拓扑结构，会引入很多单机系统没有的问题，为了解决这些问题又会引入更多的机制、协议，带来更多的问题。。。

　　在很多文章中，主要讲分布式系统分为分布式计算（computation）与分布式存储（storage）。计算与存储是相辅相成的，计算需要数据，要么来自实时数据（流数据），要么来自存储的数据；而计算的结果也是需要存储的。在操作系统中，对计算与存储有非常详尽的讨论，分布式系统只不过将这些理论推广到多个节点罢了。

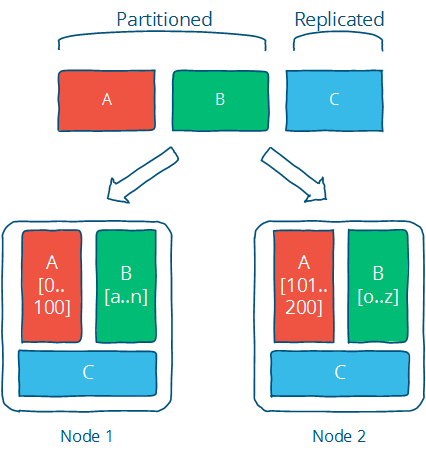
　　那么分布式系统怎么将任务分发到这些计算机节点呢，很简单的思想，分而治之，即分片（**partition）**。对于计算，那么就是对计算任务进行切换，每个节点算一些，最终汇总就行了，这就是MapReduce的思想；对于存储，更好理解一下，每个节点存一部分数据就行了。当数据规模变大的时候，Partition是唯一的选择，同时也会带来一些好处：

　　（1）提升性能和并发，操作被分发到不同的分片，相互独立

　　（2）提升系统的可用性，即使部分分片不能用，其他分片不会受到影响

　　理想的情况下，有分片就行了，但事实的情况却不大理想。原因在于，分布式系统中有大量的节点，且通过网络通信。单个节点的故障（进程crash、断电、磁盘损坏）是个小概率事件，但整个系统的故障率会随节点的增加而指数级增加，网络通信也可能出现断网、高延迟的情况。在这种一定会出现的“异常”情况下，分布式系统还是需要继续稳定的对外提供服务，即需要较强的容错性。最简单的办法，就是冗余或者复制集（**Replication**），即多个节点负责同一个任务，最为常见的就是分布式存储中，多个节点复杂存储同一份数据，以此增强可用性与可靠性。同时，Replication也会带来性能的提升，比如数据的locality可以减少用户的等待时间。

　　下面这种来自[Distributed systems for fun and profit](http://book.mixu.net/distsys/)的图形象生动说明了Partition与Replication是如何协作的。



　　Partition和Replication是解决分布式系统问题的一记组合拳，很多具体的问题都可以用这个思路去解决。但这并不是银弹，往往是为了解决一个问题，会引入更多的问题，比如为了可用性与可靠性保证，引用了冗余（复制集）。有了冗余，各个副本间的一致性问题就变得很头疼，一致性在系统的角度和用户的角度又有不同的等级划分。如果要保证强一致性，那么会影响可用性与性能，在一些应用（比如电商、搜索）是难以接受的。如果是最终一致性，那么就需要处理数据冲突的情况。CAP、FLP这些理论告诉我们，在分布式系统中，没有最佳的选择，都是需要权衡，做出最合适的选择。

**分布式系统挑战**

　　分布式系统需要大量机器协作，面临诸多的挑战：

　　第一，异构的机器与网络：

　　　　分布式系统中的机器，配置不一样，其上运行的服务也可能由不同的语言、架构实现，因此处理能力也不一样；节点间通过网络连接，而不同网络运营商提供的网络的带宽、延时、丢包率又不一样。怎么保证大家齐头并进，共同完成目标，这四个不小的挑战。

　　第二，普遍的节点故障：

　　　　虽然单个节点的故障概率较低，但节点数目达到一定规模，出故障的概率就变高了。分布式系统需要保证故障发生的时候，系统仍然是可用的，这就需要监控节点的状态，在节点故障的情况下将该节点负责的计算、存储任务转移到其他节点

　　第三，不可靠的网络：

　　　　节点间通过网络通信，而网络是不可靠的。可能的网络问题包括：网络分割、延时、丢包、乱序。

　　　　相比单机过程调用，网络通信最让人头疼的是超时：节点A向节点B发出请求，在约定的时间内没有收到节点B的响应，那么B是否处理了请求，这个是不确定的，这个不确定会带来诸多问题，最简单的，是否要重试请求，节点B会不会多次处理同一个请求。

　　总而言之，分布式的挑战来自**不确定性**，不确定计算机什么时候crash、断电，不确定磁盘什么时候损坏，不确定每次网络通信要延迟多久，也不确定通信对端是否处理了发送的消息。而分布式的规模放大了这个不确定性，不确定性是令人讨厌的，所以有诸多的分布式理论、协议来保证在这种不确定性的情况下，系统还能继续正常工作。

　　而且，很多在实际系统中出现的问题，来源于设计时的盲目乐观，觉得这个、那个应该不会出问题。[Fallacies\_of\_distributed\_computing](https://en.wikipedia.org/wiki/Fallacies_of_distributed_computing)很有意思，介绍了分布式系统新手可能的错误的假设：

The network is reliable.

Latency is zero.

Bandwidth is infinite.

The network is secure.

Topology doesn't change.

There is one administrator.

Transport cost is zero.

The network is homogeneous.

　　刘杰在《分布式系统原理介绍》中指出，处理这些异常的最佳原则是：在设计、推导、验证分布式系统的协议、流程时,最重要的工作之一就是思考在执行流程的每个步骤时一旦发生各种异常的情况下系统的处理方式及造成的影响。

**分布式系统特性与衡量标准**

　　透明性：使用分布式系统的用户并不关心系统是怎么实现的，也不关心读到的数据来自哪个节点，对用户而言，分布式系统的最高境界是用户根本感知不到这是一个分布式系统，在《[Distributed Systems Principles and Paradigms](http://barbie.uta.edu/~jli/Resources/MapReduce&Hadoop/Distributed%20Systems%20Principles%20and%20Paradigms.pdf)》一书中，作者是这么说的：

A distributed system is a collection of independent computers that appears to its users as a single coherent system.

　　可扩展性：分布式系统的根本目标就是为了处理单个计算机无法处理的任务，当任务增加的时候，分布式系统的处理能力需要随之增加。简单来说，要比较方便的通过增加机器来应对数据量的增长，同时，当任务规模缩减的时候，可以撤掉一些多余的机器，达到动态伸缩的效果

　　可用性与可靠性：一般来说，分布式系统是需要长时间甚至7\*24小时提供服务的。可用性是指系统在各种情况对外提供服务的能力，简单来说，可以通过不可用时间与正常服务时间的必知来衡量；而可靠性而是指计算结果正确、存储的数据不丢失。

　　高性能：不管是单机还是分布式系统，大家都非常关注性能。不同的系统对性能的衡量指标是不同的，最常见的：高并发，单位时间内处理的任务越多越好；低延迟：每个任务的平均时间越少越好。这个其实跟操作系统CPU的调度策略很像

　　一致性：分布式系统为了提高可用性可靠性，一般会引入冗余（复制集）。那么如何保证这些节点上的状态一致，这就是分布式系统不得不面对的一致性问题。一致性有很多等级，一致性越强，对用户越友好，但会制约系统的可用性；一致性等级越低，用户就需要兼容数据不一致的情况，但系统的可用性、并发性很高很多。

**组件、理论、协议**

　　假设这是一个对外提供服务的大型分布式系统，用户连接到系统，做一些操作，产生一些需要存储的数据，那么在这个过程中，会遇到哪些组件、理论与协议呢

**用一个请求串起来**

　　用户使用Web、APP、SDK，通过HTTP、TCP连接到系统。在分布式系统中，为了高并发、高可用，一般都是多个节点提供相同的服务。那么，第一个问题就是具体选择哪个节点来提供服务，这个就是负载均衡（load balance）。负载均衡的思想很简单，但使用非常广泛，在分布式系统、大型网站的方方面面都有使用，或者说，只要涉及到多个节点提供同质的服务，就需要负载均衡。

　　通过负载均衡找到一个节点，接下来就是真正处理用户的请求，请求有可能简单，也有可能很复杂。简单的请求，比如读取数据，那么很可能是有缓存的，即分布式缓存，如果缓存没有命中，那么需要去数据库拉取数据。对于复杂的请求，可能会调用到系统中其他的服务。

　　承上，假设服务A需要调用服务B的服务，首先两个节点需要通信，网络通信都是建立在TCP/IP协议的基础上，但是，每个应用都手写socket是一件冗杂、低效的事情，因此需要应用层的封装，因此有了HTTP、FTP等各种应用层协议。当系统愈加复杂，提供大量的http接口也是一件困难的事情。因此，有了更进一步的抽象，那就是RPC（remote produce call），是的远程调用就跟本地过程调用一样方便，屏蔽了网络通信等诸多细节，增加新的接口也更加方便。

　　一个请求可能包含诸多操作，即在服务A上做一些操作，然后在服务B上做另一些操作。比如简化版的网络购物，在订单服务上发货，在账户服务上扣款。这两个操作需要保证原子性，要么都成功，要么都不操作。这就涉及到分布式事务的问题，分布式事务是从应用层面保证一致性：某种守恒关系。

　　上面说道一个请求包含多个操作，其实就是涉及到多个服务，分布式系统中有大量的服务，每个服务又是多个节点组成。那么一个服务怎么找到另一个服务（的某个节点呢）？通信是需要地址的，怎么获取这个地址，最简单的办法就是配置文件写死，或者写入到数据库，但这些方法在节点数据巨大、节点动态增删的时候都不大方便，这个时候就需要服务注册与发现：提供服务的节点向一个协调中心注册自己的地址，使用服务的节点去协调中心拉取地址。

　　从上可以看见，协调中心提供了中心化的服务：以一组节点提供类似单点的服务，使用非常广泛，比如命令服务、分布式锁。协调中心最出名的就是chubby，zookeeper。

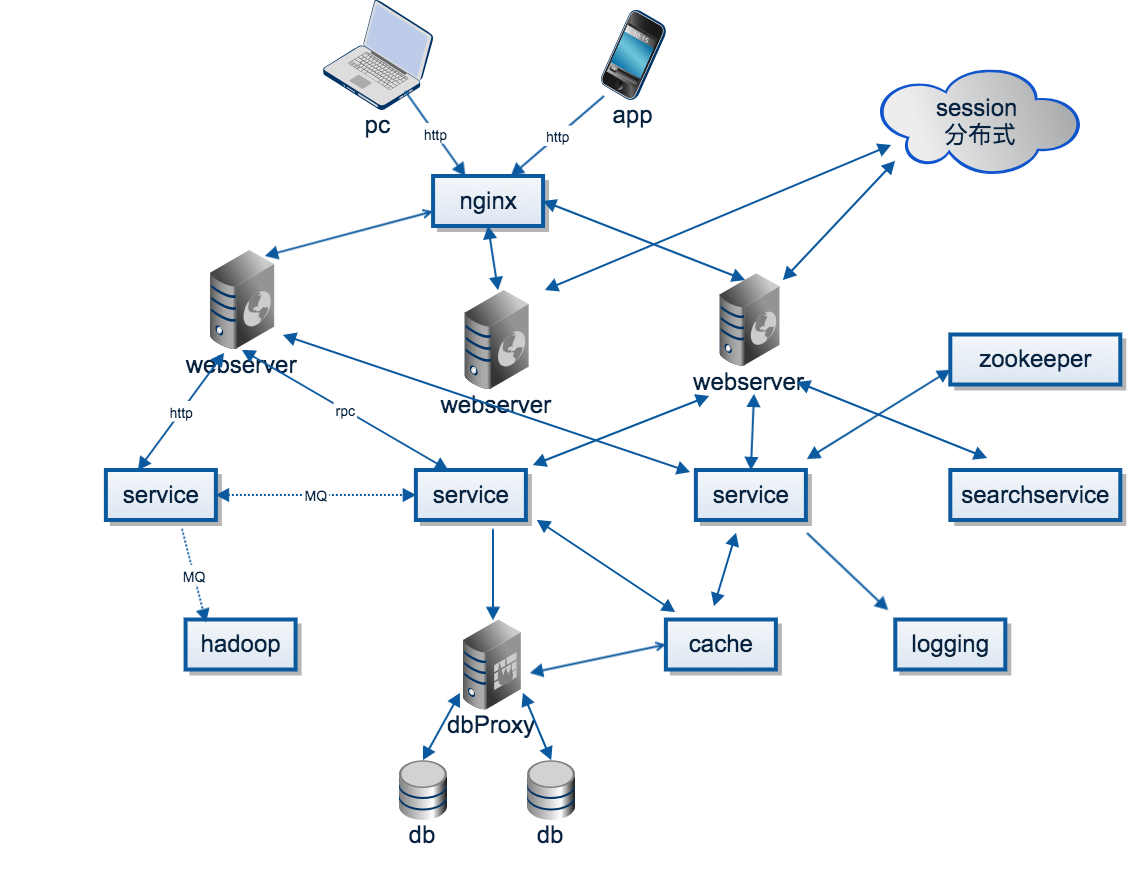
　　回到用户请求这个点，请求操作会产生一些数据、日志，通常为信息，其他一些系统可能会对这些消息感兴趣，比如个性化推荐、监控等，这里就抽象出了两个概念，消息的生产者与消费者。那么生产者怎么讲消息发送给消费者呢，RPC并不是一个很好的选择，因为RPC肯定得指定消息发给谁，但实际的情况是生产者并不清楚、也不关心谁会消费这个消息，这个时候消息队列就出马了。简单来说，生产者只用往消息队列里面发就行了，队列会将消息按主题（topic）分发给关注这个主题的消费者。消息队列起到了异步处理、应用解耦的作用。

　　上面提到，用户操作会产生一些数据，这些数据忠实记录了用户的操作习惯、喜好，是各行各业最宝贵的财富。比如各种推荐、广告投放、自动识别。这就催生了分布式计算平台，比如Hadoop，Storm等，用来处理这些海量的数据。

　　最后，用户的操作完成之后，用户的数据需要持久化，但数据量很大，大到按个节点无法存储，那么这个时候就需要分布式存储：将数据进行划分放在不同的节点上，同时，为了防止数据的丢失，每一份数据会保存多分。传统的关系型数据库是单点存储，为了在应用层透明的情况下分库分表，会引用额外的代理层。而对于NoSql，一般天然支持分布式。

**一个简化的架构图**

　　下面用一个不大精确的架构图，尽量还原分布式系统的组成部分（不过只能体现出技术，不好体现出理论）



**概念与实现**

　　那么对于上面的各种技术与理论，业界有哪些实现呢，下面进行简单罗列。

　　当然，下面的这些实现，小部分我用过，知其所以然；大部分听说过，知其然；还有一部分之前闻所未闻，分类也不一定正确，只是从其他文章抄过来的。罗列在这里，以便日后或深或浅的学习。

负载均衡：

　　　　Nginx：高性能、高并发的web服务器；功能包括负载均衡、反向代理、静态内容缓存、访问控制；工作在应用层

　　　　LVS： Linux virtual server，基于集群技术和Linux操作系统实现一个高性能、高可用的服务器；工作在网络层

webserver：

　　　　Java：Tomcat，Apache，Jboss

　　　　Python：gunicorn、uwsgi、twisted、webpy、tornado

service：

　　　　SOA、微服务、spring boot，django

容器：

　　　　docker，kubernetes

cache：

　　　　memcache、redis等

协调中心：

　　　　zookeeper、etcd等

　　　　zookeeper使用了Paxos协议Paxos是强一致性，高可用的去中心化分布式。zookeeper的使用场景非常广泛，之后细讲。

rpc框架：

　　　　grpc、dubbo、brpc

　　　　dubbo是阿里开源的Java语言开发的高性能RPC框架，在阿里系的诸多架构中，都使用了dubbo + spring boot

消息队列：

　　　　kafka、rabbitMQ、rocketMQ、QSP

　　　　消息队列的应用场景：异步处理、应用解耦、流量削锋和消息通讯

实时数据平台：

　　　　storm、akka

离线数据平台：

　　　　hadoop、spark

　　　　PS: apark、akka、kafka都是scala语言写的，看到这个语言还是很牛逼的

dbproxy：

　　　　cobar也是阿里开源的，在阿里系中使用也非常广泛，是关系型数据库的sharding + replica 代理

db：

　　　　mysql、oracle、MongoDB、HBase

搜索：

　　　　elasticsearch、solr

日志：

　　　　rsyslog、elk、flume

**总结**

　　写这篇文章，我曾在网络上搜索过“如何学习分布式系统”，但实话说，没有很认同的答案。也许，这确实是一个难以回答的问题。于是，我想自己写出一个答案，但写完这篇文章，感觉自己的回答也很混乱，也没有说清楚，不过对我自己还是有一些指导意义的，比如，理清了分布式系统中会遇到的各种技术、理论、协议，以及通过一个例子展示他们是如何协作的，接下来就是各个击破了。

　　网上的诸多回答，上来就是看各种论文，google三大件、paxos什么的，个人觉得不是很实用。更好的过程，是先有一个整体的把握，然后自己思考会有什么问题，带着问题去寻求答案，在寻求答案的时候再去看论文。

　　另外，也有很多人提到，掌握好计算机基础知识，如操作系统、计算机网络，对学习分布式系统是大有裨益的，这一点我很赞同。分布式系统解决问题的思路是早就有的，很多都是前人研究透的问题，思想都是相同的。比如函数式编程中的map reduce之于Hadoop的MapReduce，比如磁盘存储的raid之于Partition与Replication，比如IPC之于消息队列。

**references**

[Distributed systems for fun and profit](http://book.mixu.net/distsys/)

刘杰：分布式原理介绍

[Fallacies\_of\_distributed\_computing](https://en.wikipedia.org/wiki/Fallacies_of_distributed_computing)

[CMU 15-440: Distributed Systems Syllabus](http://www.cs.cmu.edu/~dga/15-440/S14/syllabus.html)

[Distributed Systems Principles and Paradigms](http://barbie.uta.edu/~jli/Resources/MapReduce&Hadoop/Distributed%20Systems%20Principles%20and%20Paradigms.pdf)

[学习分布式系统需要怎样的知识？](https://www.zhihu.com/question/23645117)