如何系统性地学习分布式系统?

前言

学习一个知识之前，我觉得比较好的方式是先理解它的来龙去脉：即这个知识产生的过程，它解决了什么问题，它是怎么样解决的，还有它引入了哪些新的问题（没有银弹），这样我们才能比较好的抓到它的脉络和关键点，不会一开始就迷失在细节中。

所以，在学习分布式系统之前，我们需要解决的第一个问题是：分布式系统解决了什么问题？

分布式系统解决了什么问题？

第一个是单机性能瓶颈导致的成本问题，由于摩尔定律失效，廉价 PC 机性能的瓶颈无法继续突破，小型机和大型机能提高更高的单机性能，但是成本太大高，一般的公司很难承受；

第二个是用户量和数据量爆炸性的增大导致的成本问题，进入互联网时代，用户量爆炸性的增大，用户产生的数据量也在爆炸性的增大，但是单个用户或者单条数据的价值其实比软件时代（比如银行用户）的价值是只低不高，所以必须寻找更经济的方案；

第三个是业务高可用的要求，对于互联网的产品来说，都要求 7 \* 24 小时提供服务，无法容忍停止服务等故障，而要提供高可用的服务，唯一的方式就是增加冗余来完成，这样就算单机系统可以支撑的服务，因为高可用的要求，也会变成一个分布式系统。

基于上面的三个原因可以看出，在互联网时代，单机系统是无法解决成本和高可用问题的，但是这两个问题对几乎对所有的公司来说都是非常关键的问题，所以，从单机系统到分布式系统是无法避免的技术大潮流。

分布式系统是怎么来解决问题的？

那么，分布式系统是怎么来解决单机系统面临的成本和高可用问题呢？

其实思路很简单，就是将一些廉价的 PC 机通过网络连接起来，共同完成工作，并且在系统中提供冗余来解决高可用的问题。

分布式系统引入了哪些新的问题？

我们来看分布式系统的定义：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。在定义中，我们可用看出，分布式系统它通过多工作节点来解决单机系统面临的成本和可用性问题，但是它引入了对分布式系统内部工作节点的协调问题。

我们经常说掌握一个知识需要理解它的前因后果，对于分布式系统来说，前因是「分布式系统解决了什么问题」，后果是「它是怎么做内部工作节点的协调」，所以我们要解决的第二个问题是：分布式系统是怎么做内部工作节点协调的？

分布式计算引入了哪些新的问题？

先从简单的情况入手，对于分布式计算（无状态）的情况，系统内部的协调需要做哪些工作：

1. 怎么样找到服务？

在分布式系统内部，会有不同的服务（角色），服务 A 怎么找到服务 B 是需要解决的问题，一般来说服务注册与发现机制是常用的思路，所以可以了解一下服务注册发现机制实现原理，并且可以思考服务注册发现是选择做成 AP 还是 CP 系统更合理（严格按 CAP 理论说，我们目前使用的大部分系统很难满足 C 或者 A 的，所以这里只是通常意义上的 AP 或者 CP）；

2. 怎么样找到实例？

找到服务后，当前的请求应该选择发往服务的哪一个实例呢？一般来说，如果同一个服务的实例都是完全对等的（无状态），那么按负载均衡策略来处理就足够（轮询、权重、hash、一致性 hash，fair 等各种策略的适用场景）；如果同一个服务的实例不是对等的（有状态），那么需要通过路由服务（元数据服务等）先确定当前要访问的请求数据做哪一个实例上，然后再进行访问。

3. 怎么样避免雪崩？

系统雪崩是指故障的由于正反馈循序导致不断扩大规则的故障。一次雪崩通常是由于整个系统中一个很小的部分出现故障于引发，进而导致系统其它部分也出现故障。比如系统中某一个服务的一个实例出现故障，导致负载均衡将该实例摘除而引起其它实例负载升高，最终导致该服务的所有实例像多米诺骨牌一样一个一个全部出现故障。

避免雪崩总体的策略比较简单，只要是两个思路，一个是快速失败和降级机制（熔断、降级、限流等），通过快速减少系统负载来避免雪崩的发生；另一个为弹性扩容机制，通过快速增加系统的服务能力来避免雪崩的发生。这个根据不同的场景可以做不同的选择，或者两个策略都使用。

一般来说，快速失败会导致部分的请求失败，如果分布式系统内部对一致性要求很高的话，快速失败会带来系统数据不一致的问题，弹性扩容会是一个比较好的选择，但是弹性扩容的实现成本和响应时间比快速失败要大得多。

4. 怎么样监控告警？

对于一个分布式系统，如果我们不能很清楚地了解内部的状态，那么高可用是没有办法完全保障的，所以对分布式系统的监控（比如接口的时延和可用性等信息），分布式追踪 Trace，模拟故障的混沌工程，以及相关的告警等机制是一定要完善的；

分布式存储引入了哪些新的问题？

接下来我们再来看分布式存储（有状态）的内部的协调是怎么做的，同时，前面介绍的分布式计算的协调方式在分布式存储中同样适用，就不再重复了：

1. 分布式系统的理论与衡权

ACID、BASE 和 CAP 理论，了解这三个主题，推荐这一篇文章以及文章后面相关的参考文献：  
英文版本：<https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed/>  
中文版本：<https://www.infoq.cn/article/cap-twelve-years-later-how-the-rules-have-changed/>

2. 怎么样做数据分片？

单机的存储能力是不可能存储所有的数据的，所以需要解决怎么将数据按一定的规则分别存储到不同的机器上，目前使用比较多的方案为：Hash、Consistent Hash 和 Range Based 分片策略，可以了解一下它们的优缺点和各自的应用场景；

3. 怎么样做数据复制？

为什么满足系统的高可用要求，需要对数据做冗余处理，目前的方案主要为：中心化方案（主从复制、一致性协议比如 Raft 和 Paxos 等）和 去中心化的方案（Quorum 和 Vector Clock）了解一下它们的优缺点和各自的应用场景，以及对系统外部表现出来的数据一致性级别（线性一致性、顺序一致性、最终一致性等）；

4. 怎么样做分布式事务？

对于分布式系统来说，要实现事务，首先需要有对并发事务进行排序的能力，这样在事务冲突的时候，确认哪个事务提供成功，哪个事务提交失败。对于单机系统来说这个完全不是问题，简单通过时间戳加序号的方式就可以实现，但是对于分布式系统来说，系统中机器的时间不能完全同步，并且单台机器序号也没用全局意义，按上面的方式说行不通的。不过整个系统选一台机器按单机的模式生产事务 ID 是可以的，同城多中心和短距离的异地多中心都没有问题，不过想做成全球分布式系统的话，那么每一次事务都要去一个节点去获取事务 ID 的成本太高（比如中国杭州到美国东部的 RTT 为 200 + ms ），Google 的 Spanner 是通过 GPS 和原子钟实现 TrueTime API 来解决这个问题从而实现全球分布式数据库的。

有了事务 ID 后，通过 2PC 或者 3PC 协议来实现分布式事务的原子性，其他部分和单机事务差别不大，就不再细说来。

进阶学习阶段

到这里，对分布式系统脉络上有了基本的概念，接下来开始进入细节学习阶段，这也是非常幸苦的阶段，对于分布式系统的理解深入与否，对细节的深入度是很重要的评价指标，毕竟魔鬼在细节。这里可以往两个方面进行系统的学习：

1. 从实践出发

研究目前比较常用的分布式系统的设计，HDFS 或者 GFS（分布式文件系统）、Kafka 和 Pulsar（分布式消息队列），Redis Cluster 和 Codis（分布式缓存），MySQL 的分库分表（传统关系型数据库的分布式方案），MongoDB 的 Replica Set 和 Sharing 机制集以及去中心化的 Cassandra（NoSQL 数据库），中心化的 TiDB 和去中心化的 CockroachDB（NewSQL），以及一些微服务框架等；

2. 从理论出发

从理论出发，研究分布式相关的论文，这里推荐一本书「Designing Data-Intensive Applications」（中文版本：数据密集型应用系统设计），先整体看书，对比较感兴趣的章节，再读一读该章节中涉及到的相关参考文献。

总结

本文从分布式系统解决的问题开始，再讨论它是怎么样来解决问题的，最后讨论了它引入了哪些新的问题，并且讨论这些新问题的解决办法，这个就是分布式系统大概的知识脉络。掌握这个知识脉络后，那么就可以从实践和理论两个角度结合起来深入细节研究分布式系统了。