# 一文看懂分布式系统的基本盘

张帅 分布式实验室 2020-02-13

本文特别写给想要学习分布式系统但是还不知道该如何下手的读者，宽泛并点到为止的介绍了我个人对于分布式系统各个方面的一些不成熟的理解，帮助读者认识到分布式系统领域的一个全景图，以便接下来寻找感兴趣的领域进行深入的学习。

学习分布式系统，需要回答以下几个问题：

1. （需求分析）分布式系统主要解决哪些问题？主要应用场景有哪些？
2. （实现方案）构建分布式系统的常见问题有哪些？解决这些问题的主流方案有哪些？
3. （技术难点）实现分布式系统的本质困难是什么？这些困难影响了那些问题？
4. （工业应用）工业上正在构建那些分布式系统？他们的发展情况如何？

特别提示，这篇文章全凭个人经验写成，有些看法和理解可能不正确，如有任何疑问，还请在评论区讨论，谢谢。

## 分布式系统的基本问题和应用场景

分布式系统主要解决以下 2 个问题：

1. Scalability，特别的，指 Scale Out
2. High Availability，特别的，指通过软件系统的设计，组合多个相对可靠性较低的设备对外提供更高的可用性

互联网时代的到来，使得我们需要解决的问题规模越来越大，以至于单机的 Capacity 增长速度难以满足需要。这里的 Capacity 可能是计算资源（CPU、内存），存储资源（硬盘），甚至是网络资源（网络 IO）。分布式系统通过在软件层面上协调多个终端共同工作来提供服务，以满足对 Capacity 的需求。其本质上和操作系统、虚拟机等技术的作用一样，都是使用软件对上层屏蔽下层的细节。

随着分布式系统规模的不断增长，人们（Google）发现大量使用通用硬件取代商业高可用硬件，可以节省大量成本。于是使用软件模拟高可用的技术应运而生，这也是分布式系统着力于解决的一大问题。

本质上，系统软件的发展就是使用软件屏蔽底层细节的过程。明确了这一思路，分布式系统的应用场景就很容易找了。先从最底层的硬件开始：

1. 计算资源（CPU）
2. 存储（硬盘、内存）
3. 通信网络（网卡、交换机）
4. 整个物理机（虚拟机）

然后上升到操作系统：

1. 调度（进程、线程）
2. 隔离（时间片共享、队列、Namespace）
3. 整个操作系统（容器）

最后到各个具体应用：Software as a Service。

特别需要注意的是，尽管都是屏蔽底层细节，但是屏蔽到什么程度仍然是值得探索的一个问题。个人认为，主张完全屏蔽下层细节，尤其是屏蔽分布式系统和单机系统这一细节的尝试基本上已经宣告失败。这方面的探索主要体现在构建分布式共享内存（Distributed Shared Memory）这一 topic 上。出于性能上的考虑，上层需要理解并一定程度上配合底层细节。因

此暴露何种细节，提供什么样的接口仍然是一个值得思考的问题。但是随着硬件和底层设施的发展，这一问题也在不断变换。

## 构建分布式系统的常见问题和解决方案

### 节点的组织

分布式系统最基础的问题在于如何将分布式节点组织起来发挥作用，这方面的讨论在论文笔记：关于 P2P 的一些综述[1]中有一个总结。大体上，分布式节点的组织方式有以下几种：

- Hybrid Decentralized：具有中心节点进行协调或索引，但是其他部分都是分布式的，例如 HDFS, BitTorrent。
- Partially Centralized：不具有中心节点，但是具有 super node，例如类似于 DNS 的结构，但是每层的上一层都是选举出来的 super node。
- Purely Decentralized：完全分布式，节点完全对等
  - Unstructured：节点不组成固定结构，类似于物理路由器，只知道附近的几个邻居节点。因此查询等操作主要依赖于 flooding 或类似于物理路由器的路由操作。
  - Structured：节点组成固定结构，例如 Chord（环），CAN（多维空间），Tapestry（超立方体）

其中 Partially Centralized 和 Purely Decentralized 没有什么本质上的区别，但是 Hybrid Decentralized 一直被诟病存在中心节点带来的性能和可用性问题。尽管从 Google 和目前各个开源系统的实践来看，Hybrid Decentralized 是一种能够以较低的成本构造（代码质量上）可靠的分布式系统的实用方法，但是随着问题规模的进一步增长，中心节点也确实成为了瓶颈（进而需要使用 Federation 的方案把中心节点再分布式化），由中心节点带来的性能问题也仍然很难解决。

分布式节点的组织必然需要通信作为支撑。对于一个存在中心节点的系统，这个问题比较好解决，只需要额外处理好中心节点的限流、延迟等问题即可。对于不存在中心节点的系统，需要解决好以下几个问题：

- Membership protocol：哪些节点是当前 Overlay network 的成员?
- Naming service：本质上是名字到位置的索引
- Message routing：如何将消息从一个位置投递到目标位置?
  - unicast
  - anycast

- groupcast
- atomic groupcast
- total ordered groupcast

**节点的协作**

由于我们需要解决的问题之一就是 Capacity 的问题，必然会出现需要取用的数据分布在多个节点的情况。在这种情况下，需要节点之间进行协作。此时，在传统并行计算领域的基础组件需要在分布式领域重新实现：

1. 临界区和锁
2. 原子性和事务

目前解决这一问题主要依赖于分布式事务，工业实现正在从 2PC 过渡到 Paxos。跨越 Shard 的 Compare-And-Swap 操作不存在。只读或者只写的分布式事务存在优化的可能。关于这部分的讨论在 CAP，ACID，我们能做什么[2]中有一个总结。

**数据的复制和一致性**

为了支持 High Availability，分布式系统一般都会将数据进行复制，但是由此可能会引起一致性问题。传统的一致性问题在并行计算领域就已经产生了，例如多核 CPU 的 Cache 就是一个典型的例子。但是一致性问题在分布式领域有着更多的困难，这一困难源于物理机之间的通信网络可靠性比物理机内部的（尤其是 CPU 核心之间，核心和内存、缓存之间）的通信低的多。关于这部分的讨论在 CAP，ACID，我们能做什么[2]中有一个总结。

## 分布式系统的本质难点

个人认为，分布式系统所有的理论难点都源于这 2 个关键点：

1. 在分布式系统中不存在全局时钟
2. 在分布式系统中通信是不可靠的（保证信息不重不漏的通信延迟是有限无界的）

由于这两点，我们不能构建事件的全序关系：

1. 不能通过时戳定序
2. 不能通过通信（在有界的延迟内）定序

上面提到的全序组播，分布式锁和事务，没有提到的节点失效检测等问题，都受这一本质问题（事件定序）的影响。

## 分布式系统的工业应用

这里只讨论比较底层的系统。

**分布式存储**

分布式存储系统是最早进入大众视野的分布式产品，以 NoSQL 为大旗抢占了传统数据库的大量市场份额。

所有存储系统的本质都是数据加上数据的索引。

最简单的分布式存储系统是 Key-Value 存储，其难点在于如何构建 Key 的索引以及所有分布式系统都需要面对的一致性问题。主流的 Key-Value 存储使用一致性哈希构建 Key 的索引，这方面的问题主要集中在如何做 scan，以及有可能需要扩展的 secondary index。关于哈希表的相关讨论见哈希表总结及其高级话题讨论[3]。也有分布式系统以树状结构构建 Key 的索引（BigTable）。

分布式存储系统的一致性问题取决于业务场景的需求，在此不进行展开。但是如何在不同的一致性级别之间平滑切换，也是学术界和工业界共同的热点。特别的，在弱一致性情况下，可能会产生数据冲突，如何提供业务友好的冲突解决方案，也是一个很好的方向。冲突解决的研究见论文笔记：[Inria RR-7506] A comprehensive study of Convergent and Commutative Replicated Data Types[4]以及论文笔记：[ICDE'18] Anna: A KVS for any scale[5]。

还有一些单机和分布式存储系统都需要面对的问题：性能优化。

1. 读多写少，读少写多等特殊场景下的性能优化
2. 随机读写和顺序读写场景下，吞吐和延迟的优化
3. 行式存储和列式存储的取舍与优化

虽然分布式存储在最初是举起 NoSQL 的大旗革了传统数据库的命，但是仍然还是需要去解决传统数据库已经解决过的问题，其中非常重要的一点就是 Schema。有了 Schema，系统就能

理解 Value 的含义，而不只是将其当做一坨 Blob。理解了 Value 的含义，才能做更多特化的事情，进一步提高性能。

目前也有一些存储系统开始尝试加上一些计算的功能，但是实现的方式还都比较初级，基本上都是将计算任务下推到数据所在的节点进行计算，再将结果汇总。个人认为存储和计算的完全分离是未来的主流方向，上面提到的实现方式的优点是延迟较低，但是分布式计算和根据元数据进行调度的系统配合起来完全能够达到这个效果，并且还有额外的好处。

此外就是特殊场景的优化。一个是 OLAP，以及 OLAP 和 OLTP 的融合。还有一个是时间序列数据的特定场景优化。

**分布式计算**

最早为人所熟知的分布式计算框架可能是 MapReduce（不算 MPI 的话）。MapReduce 的模型比较简单，很难满足业务复杂的需求，于是 Tez 和 Spark 这样的系统应运而生。实践上来看，RDD+DAG 的模型已经能够很好地表述大多数业务逻辑。

流式计算和批量计算没有本质上的区别，基本上可以认为批量计算是流式计算的一个特例。但是由于历史原因，目前批量计算框架在进行批量计算时，性能要强于流式算计框架。流式计算如果不关心数据的顺序的话，可以采用 micro batch 的方式进行处理。流式计算如果关心数据的顺序的话，问题会变得稍微复杂一些。困难在于，只从数据的角度来看的话，数据在进行划分的时候是有限无界的。但是从业务的角度来看，如果数据的延迟足够大的话，可以认为这些数据可以被抛弃（意味着不存在这样的数据），因此可以在时间限制上强行划出硬界限。从业务的角度来看，数据的延迟大概率会在一个软界限以内，可以以这个界限对数据进行划分，超出软界限的数据走旁路流程进行补救，从而将延迟大概率控制到软界限以内。批量计算可以认为是界限分明的数据进行流式计算。

仍然有一些业务要求比 Dataflow 更复杂的计算模型。最通用的模型就是用户自己写代码，可以允许任意复杂，即对托管服务的支持。目前 Kubernetes 天然支持托管服务，YARN 在这方面的支持较少，也没有很出色的通用框架。对于其他一些应用比较广泛的计算模型，例如图计算，机器学习，目前已经有了一些计算框架支持。

人们对于本地性的看法也随着时间发展发生了一些变化。最早的时候由于缺乏分布式存储，人们别无选择只能让数据和计算绑定。MapReduce 提出的另一个思想就是计算跟着数据跑，尽可能的把计算分配到数据所在的节点。这一思想其实在 HPC 中也早有应用。但是随着网络栈速度的大幅提升，人们发现本地性不再像以前一样重要（当然，仍然很重要）：

1. 计算和存储的比例很难事先确定

2. TOR 内与 POD 内网络带宽相比较不再具有压倒性的优势

另一方面，随着集群规模的扩大和迭代，人们很难保证一个集群内的机器都是同构的（之前在 HPC 的时代很容易保证），于是对于异构调度的需求变得逐渐强烈起来，并且压倒了本地性的需求。于是最终调度从计算框架中分离出来作为一个相对独立的中间件存在。

目前看来，分布式计算领域主流计算模型已经有了可用的计算框架支持，后续的问题主要在于如何进一步提高性能（吞吐和延迟两方面）。个人认为近几年的性能优化主要还是把数据库领域中已有的功能拿回来，我们应该更多的关注 VLDB 这样的数据库期刊。另外一些优化的点则在于分布式系统特有的问题，尤其是系统的组件可靠性比传统 HPC 低造成的问题。

**分布式调度**

由于大量使用通用硬件代替 HPC，集群内的机器开始出现异构的情况，并且用户的任务也是异构的，于是对于调度的要求提高了很多。分布式调度主要有以下几个发展方向：

1. 精细化调度
2. 特殊场景的调度
3. 可插拔的灵活调度

精细化调度体现在以下几个方面：

1. 调度的粒度变得更细（整机 →Slot→Fine Grain）
2. 需要调度的资源类型和维度都在增长（可伸缩 / 不可伸缩，CPU/MEM/GPU/IO/……）
3. 调度的物理机情况变得更加复杂（资源异构，亲和性，条件限制……）
4. 考虑目标机器状态的调度（机会调度，超售……）
5. 考虑全局策略的调度

特殊场景的调度有些是可以在性能上进行极大的优化，有些是有特殊的限制需求，例如 Gang Scheduling，有 Deadline 要求的调度等等。

目前分布式集群的使用方式也和 HPC 大有不同，不再是一个任务跑完了再跑另一个任务了，而是多种任务一起去跑。在这种情况下，不同的任务对于调度的需求又有不同，因此需要可插拔的灵活调度机制。

总体而言，调度的发展趋势更多的要求全局信息。目前由于没有成熟的分布式数据库方案的支持，一般选择中心化全局信息的方案，这反过来又限制了集群的规模。为了解决集群规模增长的问题，一般使用 Sharding 和 Facade 的方式去做，但是也有难以解决的问题。

这里插一段个人思考，调度是否适合采用乐观并发？如果调度如果采用乐观并发的话，对于一致性的需求就降低了，这样的话不使用分布式数据库也能解决这个问题。

**其他中间件**

除了上面提到的比较大块的内容以外，也有专注于解决具体细节问题的方向。

首先要解决的就是节点的组织（含一致性哈希等方式）和消息路由（含 Sharding）的问题，这方面比较知名的中间件有 Consul 和 Akka。节点组织起来之后也有可能会失效，因此需要失效检测的中间件（Failure Detector），但是这在分布式系统中是一个困难问题。然后有的人致力于解决可靠传输的问题，这方面的发展趋势是将协议栈倒置，让传输层而非应用层称为最上层，例如 Kafka。另一个趋势是 Message Broker 越来越向通用存储发展，两者的界限越来越不分明，例如 Kubernetes 使用 etcd 来做"消息传递"。锁，分布式事务，选主，这三个是一回事，目前稳定的实现是 Raft 的各个变种，趋势是使用 Paxos，因为优化的余地更大（Raft 相当于是 Paxos 的串行版本）。

分布式带来的另一个问题是 DEBUG 的难度进一步上升，因为一个用户请求会分散到不同的机器上。为此我们使用 3 种不同的方法来解决这一问题（Observability 3 ways: logging metrics and tracing[6]）：

1. 收集日志，日志包括一台机器上最为详细的细节信息
2. 收集 Metrics，Metrics 由于只是数因此方便做统计
3. 做 Tracing，将一个用户请求的全过程串起来

除此之外还有很多小的领域，不在此一一介绍。

相关链接：

1. https://zhuanlan.zhihu.com/p/34323809
2. https://zhuanlan.zhihu.com/p/37076900
3. https://zhuanlan.zhihu.com/p/35673745
4. https://zhuanlan.zhihu.com/p/33912913
5. https://zhuanlan.zhihu.com/p/35426881
6. https://speakerdeck.com/adriancole/observability-3-ways-logging-metrics-and-tracing

原文链接：https://zhuanlan.zhihu.com/p/60916031

## Kubernetes入门与实战培训

**Kubernetes入门与实战培训将于2020年2月28日在北京开课，3天时间带你系统掌握Kubernetes，学习效果不好可以继续学习。**本次培训包括：Docker基础、容器技术、Docker镜像、数据共享与持久化、Docker实践、Kubernetes基础、Pod基础与进阶、常用对象操作、服务发现、Helm、Kubernetes核心组件原理分析、Kubernetes服务质量保证、调度详解与应用场景、网络、基于Kubernetes的CI/CD、基于Kubernetes的配置管理等等，点击下方图片或者阅读原文链接查看详情。


阅读原文

喜欢此内容的人还喜欢

关于 Go 语言，你不得不知的并发模式 | 留言送书

分布式实验室

跟你们说个事

跟你们说个事

服装搭配师miuo

---

深度 | 好牌打得稀烂，传统美食跌落神坛的秘密

一个坏土豆