

16 | InfluxDB企业版一致性实现剖析：他山之石，可以攻玉

2020-03-18 韩健

分布式协议与算法实战

[进入课程 >](#)



讲述：于航

时长 14:34 大小 13.34M



你好，我是韩健。

学习了前面 15 讲的内容后，我们了解了很多常用的理论和算法（比如 CAP 定理、Raft 算法等）。是不是理解了这些内容，就能够游刃有余地处理实际系统的问题了呢？

在我看来，还远远不够，因为理论和实践的中间是存在鸿沟的，比如，你可能有这样的感受，提到编程语言的语法或者分布式算法的论文，你说起来头头是道，但遇到实际系统时，还是无法写程序，开发分布式系统。



而我常说，实战是学习的最终目的。为了帮你更好地掌握前面的理论和算法，接下来，我用 5 讲的时间，分别以 InfluxDB 企业版一致性实现、Hashicorp Raft、KV 系统开发实战为


例，带你了解如何在实战中使用技术，掌握分布式的实战能力。

今天这一讲，我就以 InfluxDB 企业版为例，带你看一看系统是如何实现一致性的。有的同学可能会问了：为什么是 InfluxDB 企业版呢？因为它是排名第一的时序数据库，相比其他分布式系统（比如 KV 存储），时序数据库更加复杂，因为我们要分别设计 2 个完全不一样的一致性模型。当你理解了这样一个复杂的系统实现后，就能更加得心应手地处理简单系统的问题了。

那么为了帮你达到这个目的。我会先介绍一下时序数据库的背景知识，因为技术是用来解决实际场景的问题的，正如我之前常说的“要根据场景特点，权衡折中来设计系统”。所以当你了解了这些背景知识后，就能更好的理解为什么要这么设计了。

什么是时序数据库？

你可以这么理解，时序数据库，就是存储时序数据的数据库，就像 MySQL 是存储关系型数据的数据库。而时序数据，就是按照时间顺序记录系统、设备状态变化的数据，比如 CPU 利用率、某一时间的环境温度等，就像下面的样子：

 复制代码

```
1 > insert cpu_usage,host=server01,location=cn-sz user=23.0,system=57.0
2 > select * from cpu_usage
3 name: cpu_usage
4 time                host      location system user
5 ----              -
6 1557834774258860710 server01 cn-sz    55     25
7 >
```

在我看来，时序数据最大的特点是数据量很大，可以不夸张地说是海量。时序数据主要来自监控（监控被称为业务之眼），而且在不影响业务运行的前提下，监控埋点是越多越好，这样才能及时发现问题、复盘故障。

那么作为时序数据库，InfluxDB 企业版的架构是什么样子呢？

你可能已经了解过，它是由 META 节点和 DATA 节点 2 个逻辑单元组成的，而且这两个节点是 2 个单独的程序。那你也许会问了，为什么不能合成到一个程序呢？答案是场景不同。

META 节点存放的是系统运行的关键元信息，比如数据库 (Database)、表 (Measurement)、保留策略 (Retention policy) 等。它的特点是一致性敏感，但读写访问量不高，需要一定的容错能力。

DATA 节点存放的是具体的时序数据。它有这样几个特点：最终一致性、面向业务、性能越高越好，除了容错，还需要实现水平扩展，扩展集群的读写性能。

我想说的是，对于 META 节点来说，节点数的多少代表的是容错能力，一般 3 个节点就可以了，因为从实际系统运行观察看，能容忍一个节点故障就可以了。但对 DATA 节点而言，节点数的多少则代表了读写性能，一般而言，在一定数量以内（比如 10 个节点）越多越好，因为节点数越多，读写性能也越高，但节点数量太多也不行，因为查询时就会出现访问节点数过多而延迟大的问题。

所以，基于不同场景特点的考虑，2 个单独程序更合适。如果 META 节点和 DATA 节点合并为一个程序，因读写性能需要，设计了一个 10 节点的 DATA 节点集群，这就意味着 META 节点集群 (Raft 集群) 也是 10 个节点。在学了 Raft 算法之后，你应该知道，这时就会出现消息数多、日志提交慢的问题，肯定不行了。（对 Raft 日志复制不了解的同学，可以回顾一下 [🔗08 讲](#)）

现在你了解时序数据库，以及 InfluxDB 企业版的 META 节点和 DATA 节点了吧？那么如何实现 META 节点和 DATA 节点的一致性呢？

如何实现 META 节点一致性？

你可以这样想象一下，META 节点存放的是系统运行的关键元信息，那么当写操作发生后，就要立即读取到最新的数据。比如，创建了数据库 “telegraf”，如果有的 DATA 节点不能读取到这个最新信息，那就会导致相关的时序数据写失败，肯定不行。

所以，META 节点需要强一致性，实现 CAP 中的 CP 模型（对 CAP 理论不熟悉的同学，可以先回顾下 [🔗02 讲](#)）。

那么，InfluxDB 企业版是如何实现的呢？

因为 InfluxDB 企业版是闭源的商业软件，通过 [🔗官方文档](#)，我们可以知道它使用 Raft 算法实现 META 节点的一致性（一般推荐 3 节点的集群配置）。那么说完 META 节点的一致

性实现之后，我接着说一说 DATA 节点的一致性实现。

如何实现 DATA 节点一致性？

我们刚刚提到，DATA 节点存放的是具体的时序数据，对一致性要求不高，实现最终一致性就可以了。但是，DATA 节点也在同时作为接入层直接面向业务，考虑到时序数据的量很大，要实现水平扩展，所以必须要选用 CAP 中的 AP 模型，因为 AP 模型不像 CP 模型那样采用一个算法（比如 Raft 算法）就可以实现了，也就是说，AP 模型更复杂，具体有这样几个实现步骤。

自定义副本数

首先，你需要考虑冗余备份，也就是同一份数据可能需要设置为多个副本，当部分节点出问题，系统仍然能读写数据，正常运行。

那么，该如何设置副本呢？答案是实现自定义副本数。

关于自定义副本数的实现，我们在 [🔗12 讲](#)介绍了，在这里就不啰嗦了。不过，我想补充一点，相比 Raft 算法节点和副本必须一一对应，也就是说，集群中有多少个节点就必须有多少个副本，你看，自定义副本数，是不是更灵活呢？

学到这里，有同学可能已经想到了，当集群支持多副本时，必然会出现一个节点写远程节点时，RPC 通讯失败的情况，那么怎么处理这个问题呢？

Hinted-handoff

我想说的是，一个节点接收到写请求时，需要将写请求中的数据转发一份到其他副本所在的节点，那么在这个过程中，远程 RPC 通讯是可能会失败的，比如网络不通了，目标节点宕机了，等等，就像下图的样子。

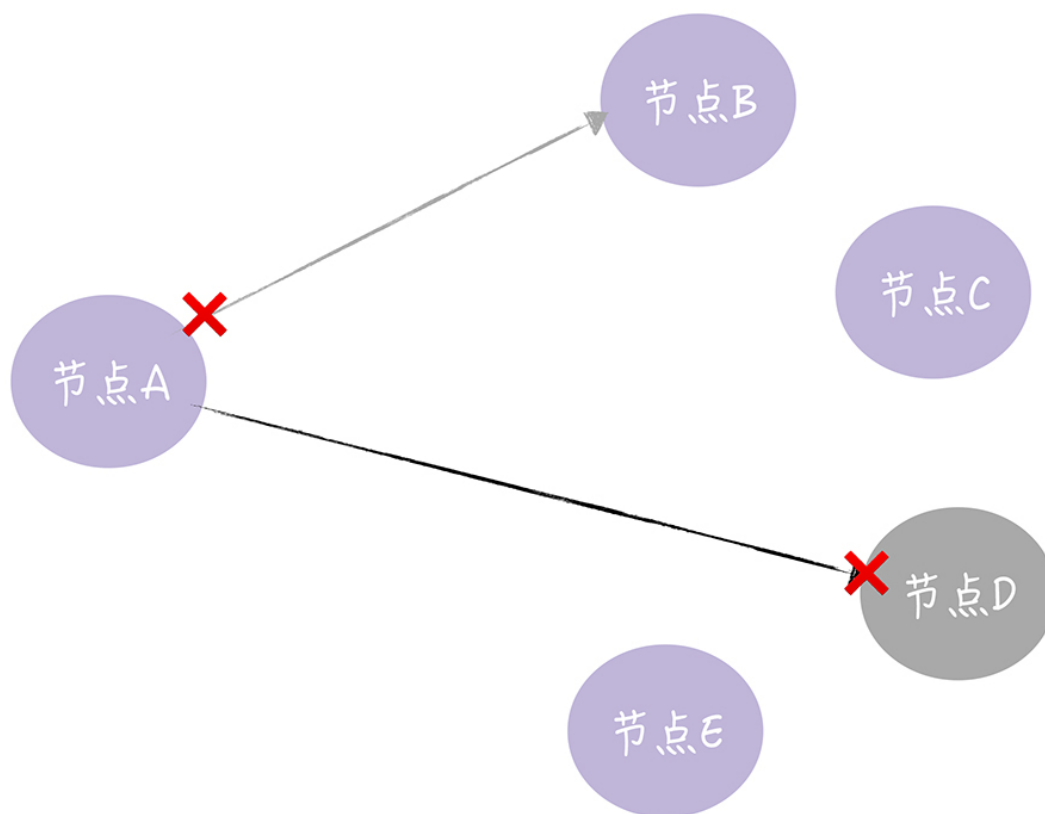


图1

那么如何处理这种情况呢？答案是实现 Hinted-handoff。在 InfluxDB 企业版中，Hinted-handoff 是这样实现的：

写失败的请求，会缓存到本地硬盘上；

周期性地尝试重传；

相关参数信息，比如缓存空间大小 (max-size)、缓存周期 (max-age)、尝试间隔 (retry-interval) 等，是可配置的。

在这里我想补充一点，除了网络故障、节点故障外，在实际场景中，临时的突发流量也会导致系统过载，出现 RPC 通讯失败的情况，这时也需要 Hinted-handoff 能力。

虽然 Hinted-handoff 可以通过重传的方式来处理数据不一致的问题，但当写失败请求的数据大于本地缓存空间时，比如某个节点长期故障，写请求的数据还是会丢失的，最终的节点的数据还是不一致的，那么怎么实现数据的最终一致性呢？答案是反熵。

反熵

需要你注意的是，时序数据虽然一致性不敏感，能容忍短暂的不一致，但如果查询的数据长期不一致的话，肯定就不行了，因为这样就会出现 “Flapping Dashboard” 的现象，也就

是说向不同节点查询数据，生成的仪表盘视图不一样，就像图 2 和图 3 的样子。

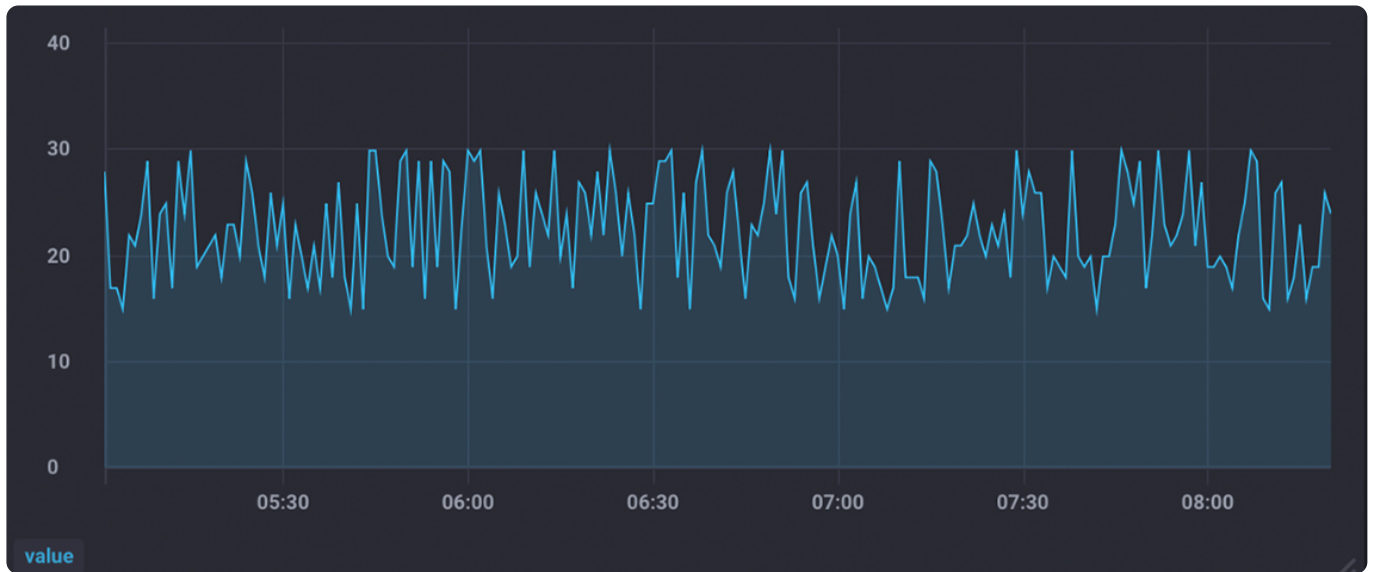


图2

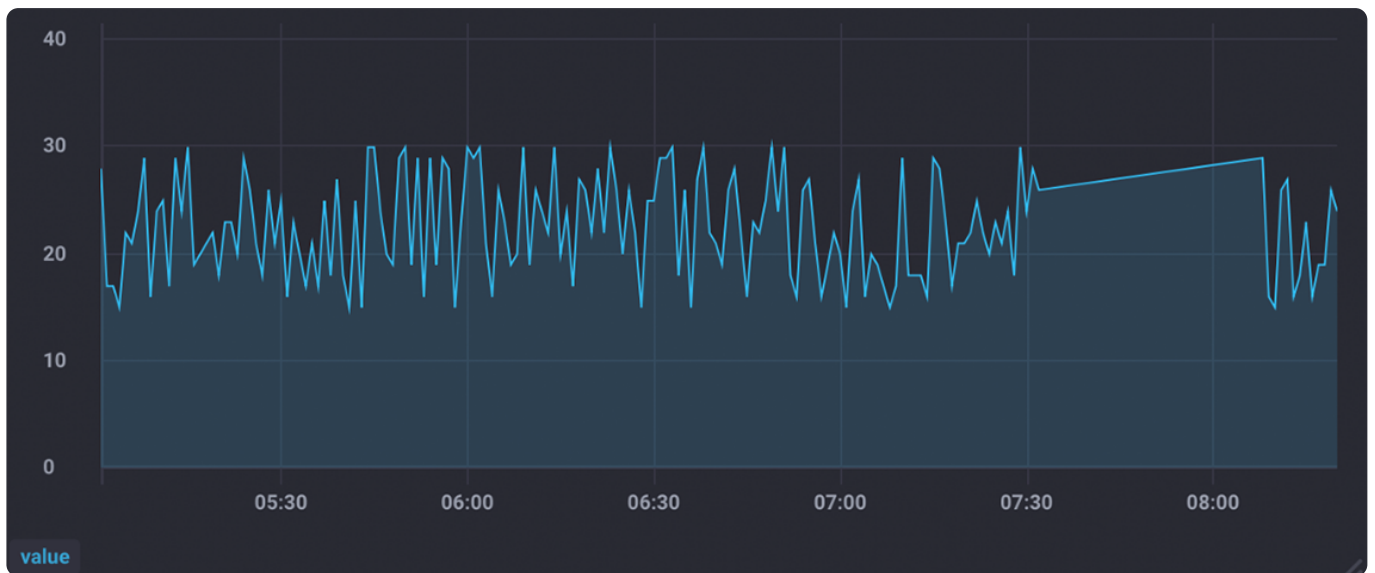


图3

从上面的 2 个监控视图中你可以看到，同一份数据，查询不同的节点，生成的视图是不一样的。那么，如何实现最终一致性呢？

答案就是咱们刚刚说的反熵，而我在 [@11 讲](#) 以自研 InfluxDB 系统为例介绍过反熵的实现，InfluxDB 企业版类似，所以在这里就不啰嗦了。

不过有的同学可能会存在这样的疑问，实现反熵是以什么为准来修复数据的不一致呢？我想说的是，时序数据像日志数据一样，创建后就不会再修改了，一直存放在那里，直到被删除。

所以，数据副本之间的数据不一致，是因为数据写失败导致数据丢失了，也就是说，存在的都是合理的，缺失的就是需要修复的。这时我们可以采用两两对比、添加缺失数据的方式，来修复各数据副本的不一致了。

Quorum NWR

最后，有同学可能会说了，我要在公司官网上展示的监控数据的仪表板（Dashboard），是不能容忍视图不一致的情况的，也就是无法容忍任何“Flapping Dashboard”的现象。那么怎么办呢？这时我们就要实现强一致性（Werner Vogels 提到的强一致性），也就是每次读操作都要能读取最新数据，不能读到旧数据。

那么在一个 AP 型的分布式系统中，如何实现强一致性呢？

答案是实现 Quorum NWR。同样，关于 Quorum NWR 的实现，我们在 12 讲已介绍，在这里也就不啰嗦了。

最后我想说的是，你可以看到，实现 AP 型分布式系统，比实现 CP 型分布式要复杂的。另外，通过上面的内容学习，我希望能注意到，技术是用来解决场景需求的，没有十全十美的技术，在实际工作中，需要我们深入研究场景特点，提炼场景需求，然后根据场景特点权衡折中，设计出适合该场景特点的分布式系统。

内容小结

本节课我主要带你了解时序数据库、META 节点一致性的实现、DATA 节点一致性的实现。以一个复杂的实际系统为例，带你将前面学习到的理论串联起来，让你知道它们如何在实际场景中使用。我希望你明确的重点如下：

1. CAP 理论是一把尺子，能辅助我们分析问题、总结归纳问题，指导我们如何做妥协折中。所以，我建议你在实践中多研究多思考，一定不能认为某某技术“真香”，十全十美了，要根据场景特点活学活用技术。
2. 通过 Raft 算法，我们能实现强一致性的分布式系统，能保证写操作完成后，后续所有的读操作，都能读取到最新的数据。
3. 通过自定义副本数、Hinted-handoff、反熵、Quorum NWR 等技术，我们能实现 AP 型分布式系统，还能通过水平扩展，高效扩展集群的读写能力。

最后，我想再强调下，技术是用来解决场景的需求的，只有当你吃透技术，深刻理解场景的需求，才能开发出适合这个场景的分布式系统。另外我还想让你知道的是，InfluxDB 企业版一年的 License 费高达 1.5 万美刀，为什么它值这个价钱？就是因为技术带来的高性能和成本优势。比如：

相比 OpenTSDB，InfluxDB 的写性能是它的 9.96 倍，存储效率是它的 8.69 倍，查询效率是它的 7.38 倍。

相比 Graphite，InfluxDB 的写性能是它的 12 倍，存储效率是 6.3 倍，查询效率是 9 倍。

在这里我想说的是，数倍或者数量级的性能优势其实就是钱，而且业务规模越大，省钱效果越突出。

另外我想说的是，尽管 influxdb-comparisons 的测试比较贴近实际场景，比如它的 DevOps 测试模型，与我们观察到常见的实际场景是一致的。但从实际效果看，InfluxDB 的优势更加明显，成本优势更加突出。因为传统的时序数据库不仅仅是性能低，而且在海量数据场景下，接入和查询的痛点突出。为了缓解这些痛点，引入和堆砌了更多的开源软件。比如：

往往需要引入 Kafka 来缓解，因突发接入流量导致的丢数据问题；

需要引入 Storm、Flink 来缓解，时序数据库计算性能差的问题；

需要做热数据的内存缓存，来解决查询超时的问题。

所以在实施中，除了原有的时序数据库会被替换掉，还有大量的开源软件会被省掉，成本优势突出。在这里我想说的是，从实际实施看（自研 InfluxDB 系统），性能优势和成本优势也是符合这个预期的。

最后我想说的是，我反对堆砌开源软件，建议谨慎引入 Kafka 等缓存中间件。老话说，在计算机中，任何问题都可以通过引入一个中间层来解决。这句话是正确的，但背后的成本是不容忽视的，尤其是在海量系统中。**我的建议是直面问题，通过技术手段在代码和架构层面解决它，而不是引入和堆砌更多的开源软件。**其实，InfluxDB 团队也是这么做，比如他们两次重构存储引擎。

课堂思考

我提到没有十全十美的技术，而是需要根据场景特点，权衡折中，设计出适合场景特点的分布式系统。那么你试着思考一下，假设有这样一个场景，一个存储系统，访问它的写请求不多（比如 1K QPS），但访问它的读请求很多（比如 1M QPS），而且客户端查询时，对数据的一致性敏感，也就是需要实现强一致性，那么我们该如何设计这个系统呢？为什么呢？欢迎在留言区分享你的看法，与我一同讨论。

最后，感谢你的阅读，如果这篇文章让你有所收获，也欢迎你将它分享给更多的朋友。

课程学习计划

关注极客时间服务号 每日学习签到

月领 25+ 极客币

【点击】保存图片，打开【微信】扫码>>>



© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 15 | ZAB协议：如何实现操作的顺序性？

下一篇 17 | Hashicorp Raft（一）：如何跨过理论和代码之间的鸿沟？

精选留言 (10)

写留言



每天晒白牙

2020-03-18

感觉自己对这些知识理解的还是不够，更不能进行实战应用，还得好好学学。

对于思考题，首先要求强一致性，读多写少，那是不是可以像 META 节点一样，采用 Raft 算法实现强一致性。但这样对性能可能就有影响了，不过这个 KV 系统是读多写少，应该也可以...

展开 ▾



👍 2



Geek_3894f9

2020-03-18

课后思考题，答案是QNWR，Wn，R1。wn是因为对写入的时间要求不高，r1是因为可以读取任意一节点，读性能好。

展开 ▾



👍 1



Fs

2020-03-23

AP型的实现大框架是类似的，influxDB的介绍和Cassandra的实现非常相似。那么支持时序这一特点，influxDB有什么不一样的设计呢

展开 ▾



Scream!

2020-03-22

多来点实战案例

展开 ▾



唔多志

2020-03-19

懵懵懂懂，似懂非懂，还是要多学习。

展开 ▾



羽翼1982

2020-03-18

在AP的系统中，使用Quorum NWR理论

如果写少读多，假设有3个副本，可以将策略调整为3个副本写成功才返回，读则可以只读取一个副本的内容；如果网络不稳定，3副本写成功到时失败率高，99线的延迟大，可以退一步使用2写2读的策略；不过很少在实际系统上看到类似3副本，允许1副本写不成功时缓存本地等待Hinted-handoff后续同步这样的策略，InfluxDB中不太清楚，至少Cassandr...

展开 ▾



hello

2020-03-18

给老师大大的赞，对实战部分很是期待，我现在就希望快点更新！



小晏子

2020-03-18

课后思考：这个里面有几个点是设计这个系统的关键，读请求（百万级）远大于写请求（千级），要求读的强一致性。

因为写请求很低，可以仿照influxDB的设计，分成meta节点和data节点，meta节点使用cp模型，使用raft算法，data节点使用ap模型，使用quorum NWR和反熵算法保证强一致性，因为写少，所以只要少量meta节点即可满足要求，比如三个，对于大量读请求，dat...
展开 ∨



pedro

2020-03-18

前面的十几讲都在为这一讲做铺垫，快更新，看看后面的实战部分 😊



夜空中最亮的星 (华仔...)

2020-03-18

喜欢案例，让案例来的更猛烈些吧

展开 ∨

