02 | 性能调优的本质：调优的手段五花八门，该从哪里入手?

2021-03-17 吴磊

Spark性能调优实战 进入课程 >

讲述：吴磊
时长 14:02 大小 12.87M

你好，我是吴磊。

上节课，我们探讨了性能调优的必要性，结论是：尽管 Spark 自身运行高效，但作为开发者，我们仍然需要对应用进行性能调优。

那么问题来了，性能调优该怎么做呢？面对成百上千行应用代码、近百个 Spark 配置项，我们该从哪里入手呢？我认为，要想弄清性能调优怎么入手，必须先得搞明白性能调优的本质是什么。

所以今天这节课，咱们就从一个先入为主的调优反例入手，带你一起探讨并归纳性能调优的本质是什么，最终帮你建立起系统化的性能调优方法论。

先入为主的反例

在典型的 ETL 场景中，我们经常需要对数据进行各式各样的转换，有的时候，因为业务需求太复杂，我们往往还需要自定义 UDF（User Defined Functions）来实现特定的转换逻辑。但是，无论是 Databricks 的官方博客，还是网上浩如烟海的 Spark 技术文章，都警告我们尽量不要自定义 UDF 来实现业务逻辑，要尽可能地使用 Spark 内置的 SQL functions。

在日常的工作中，我发现这些警告被反复地用于 Code review 中，Code reviewer 在审查代码的时候，一旦遇到自定义的 UDF，就提示开发的同学用 SQL functions 去重写业务逻辑，这几乎成了一种条件反射。

甚至，开发的同学也觉得非常有道理。于是，他们花费大量时间用 SQL functions 重构业务代码。但遗憾的是，这么做之后 ETL 作业端到端的执行性能并没有什么显著的提升。这种情况就是所谓的投入时间与产出不成正比的窘境：**调优的时间没少花，却没啥效果。**

之所以会出现这种情况，我觉得主要原因在于 Code reviewer 对于性能调优的理解还停留在照本宣科的层次，没有形成系统化的方法论。要建立系统化的方法论，我们就必须去探究性能调优的本质到底是什么。否则，开发者就像是掉进迷宫的小仓鼠，斗志昂扬地横冲直撞，但就是找不到出口。

既然性能调优的本质这么重要，那么它到底是什么呢？

性能调优的本质

探究任何一个事物的本质，都离不开从个例观察、现象归因到总结归纳的过程。

咱们不妨先来分析分析上面的反例，为什么用 SQL functions 重构 UDF 并没有像书本上说的那么奏效？

是因为这条建议本身就不对吗？肯定不是。通过对比查询计划，我们能够明显看到 UDF 与 SQL functions 的区别。Spark SQL 的 Catalyst Optimizer 能够明确感知 SQL functions 每一步在做什么，因此有足够的优化空间。相反，UDF 里面封装的计算逻辑对于 Catalyst Optimizer 来说就是个黑盒子，除了把 UDF 塞到闭包里面去，也没什么其他工作可做的。

那么，是因为 UDF 相比 SQL functions 其实并没有性能开销吗？也不是。我们可以做个性能单元测试，从你的 ETL 应用中随意挑出一个自定义的 UDF，尝试用 SQL functions 去重写，然后准备单元测试数据，最后在单机环境下对比两种不同实现的运行时间。通常情况下，UDF 实现相比 SQL functions 会慢 3% 到 5% 不等。所以你看，UDF 的性能开销还是有的。

既然如此，我们在使用 SQL functions 优化 UDF 的时候，为什么没有显著提升端到端 ETL 应用的整体性能呢？

根据木桶理论，最短的木板决定了木桶的容量，因此，对于一只有短板的木桶，其他木板调节得再高也无济于事，最短的木板才是木桶容量的瓶颈。对于 ETL 应用这支木桶来说，UDF 到 SQL functions 的调优之所以对执行性能的影响微乎其微，根本原因在于它不是那块最短的木板。换句话说，ETL 应用端到端执行性能的瓶颈不是开发者自定义的 UDF。

结合上面的分析，**性能调优的本质**我们可以归纳为 4 点。

1. 性能调优不是一锤子买卖，补齐一个短板，其他板子可能会成为新的短板。因此，它是一个动态、持续不断的过程。
2. 性能调优的手段和方法是否高效，取决于它针对的是木桶的长板还是瓶颈。针对瓶颈，事半功倍；针对长板，事倍功半。
3. 性能调优的方法和技巧，没有一定之规，也不是一成不变，随着木桶短板的此消彼长需要相应的动态切换。
4. 性能调优的过程收敛于一种所有木板齐平、没有瓶颈的状态。

基于对性能调优本质的理解，我们就能很好地解释日常工作中一些常见的现象。比如，我们经常发现，明明是同样一种调优方法，在你那儿好使，在我这儿却不好使。再比如，从网上看到某位 Spark 大神呕心沥血总结的调优心得，你拿过来一条一条地试，却发现效果并没有博客中说的那么显著。这也并不意味着那位大神的最佳实践都是空谈，更可能是他总结的那些点并没有触达到你的瓶颈。

定位性能瓶颈的途径有哪些？

你可能会问：“既然调优的关键在于瓶颈，我该如何定位性能瓶颈呢？”我觉得，至少有两种途径：**一是先验的专家经验，二是后验的运行时诊断。**下面，我们一一来看。

所谓专家经验是指在代码开发阶段、或是在代码 Review 阶段，凭借过往的实战经验就能够大致判断哪里可能是性能瓶颈。显然，这样的专家并不好找，一名开发者要经过大量的积累才能成为专家，如果你身边有这样的人，一定不要放过他！

但你也可能会说：“我身边要是有这样的专家，就不用来订阅这个专栏了。”没关系，咱们还有第二种途径：运行时诊断。

运行时诊断的手段和方法应该说应有尽有、不一而足。比如：对于任务的执行情况，Spark UI 提供了丰富的可视化面板，来展示 DAG、Stages 划分、执行计划、Executor 负载均衡情况、GC 时间、内存缓存消耗等等详尽的运行时状态数据；对于硬件资源消耗，开发者可以利用 Ganglia 或者系统级监控工具，如 top、vmstat、iostat、iftop 等等来实时监测硬件的资源利用率；特别地，针对 GC 开销，开发者可以将 GC log 导入到 JVM 可视化工具，从而一览任务执行过程中 GC 的频率和幅度。

对于这两种定位性能瓶颈的途径来说，专家就好比是老中医，经验丰富、火眼金睛，往往通过望、闻、问、切就能一眼定位到瓶颈所在；而运行时诊断更像是西医中的各种检测仪器和设备，如听诊器、X 光、CT 扫描仪，需要通过量化的指标才能迅速定位问题。二者并无优劣之分，反而是结合起来的效率更高。就像一名医术高超、经验丰富的大夫，手里拿着你的血液化验和 B 超结果，对于病灶的判定自然更有把握。

你可能会说：“尽管有这两种途径，但是就瓶颈定位来说，我还是不知道具体从哪里切入呀！”结合过往的调优经验，我认为，**从硬件资源消耗的角度切入，往往是个不错的选择**。我们都知道，从硬件的角度出发，计算负载划分为计算密集型、内存密集型和 IO 密集型。如果我们能够明确手中的应用属于哪种类型，自然能够缩小搜索范围，从而更容易锁定性能瓶颈。

不过，在实际开发中，并不是所有负载都能明确划分出资源密集类型。比如说，Shuffle、数据关联这些数据分析领域中的典型场景，它们对 CPU、内存、磁盘与网络的要求都很高，任何一个环节不给力都有可能形成瓶颈。因此，**就性能瓶颈定位来说，除了从硬件资源的视角出发，我们还需要关注典型场景**。

性能调优的方法与手段

假设，通过运行时诊断我们成功地定位到了性能瓶颈所在，那么，具体该如何调优呢？Spark 性能调优的方法和手段丰富而又庞杂，如果一条一条地去罗列，不仅看上去让人昏

昏欲睡，更不利于形成系统化的调优方法论。就像医生在治疗某个病症的时候，往往会结合内服、外用甚至是外科手术，这样多管齐下的方式来达到药到病除的目的。

因此，在我看来， Spark 的性能调优可以**从应用代码和 Spark 配置项**这 2 个层面展开。

应用代码层面指的是从代码开发的角度，我们该如何取舍，如何以性能为导向进行应用开发。在上一讲中我们已经做过对比，哪怕是两份功能完全一致的代码，在性能上也会有很大的差异。因此我们需要知道，**开发阶段都有哪些常规操作、常见误区，从而尽量避免在代码中留下性能隐患**。

Spark 配置项想必你并不陌生，**Spark 官网上罗列了近百个配置项，看得人眼花缭乱，但并不是所有的配置项都和性能调优息息相关，因此我们需要对它们进行甄别、归类**。

总的来说，在日常的调优工作中，我们往往从应用代码和 Spark 配置项这 2 个层面出发。所谓：“问题从代码中来，解决问题要到代码中去”，在应用代码层面进行调优，其实就是一个捕捉和移除性能 BUG 的过程。Spark 配置项则给予了开发者极大的灵活度，允许开发者通过配置项来调整不同硬件的资源利用率，从而适配应用的运行时负载。

对于应用代码和 Spark 配置项层面的调优方法与技巧，以及这些技巧在典型场景和硬件资源利用率调控中的综合运用，我会在《性能篇》逐一展开讲解，从不同层面、不同场景、不同视角出发，归纳出一套调优的方法与心得，力图让你能够按图索骥、有章可循地去开展性能调优。

性能调优的终结

性能调优的本质告诉我们：性能调优是一个动态、持续不断的过程，在这个过程中，调优的手段需要随着瓶颈的此消彼长而相应地切换。那么问题来了，性能调优到底什么时候是个头儿呢？就算是持续不断地，也总得有个收敛条件吧？总不能一直这么无限循环下去。

在我看来，**性能调优的最终目的，是在所有参与计算的硬件资源之间寻求协同与平衡，让硬件资源达到一种平衡、无瓶颈的状态**。

我们以大数据服务公司 Qubole 的案例为例，他们最近在 Spark 上集成机器学习框架 XGBoost 来进行模型训练，在相同的硬件资源、相同的数据源、相同的计算任务中对比不同配置下的执行性能。

从下表中我们就可以清楚地看到，执行性能最好的训练任务并不是那些把 CPU 利用率压榨到 100%，以及把内存设置到最大的配置组合，而是那些硬件资源配置最均衡的计算任务。

运行时间	CPU负载	CPU/任务	CPU/Executor	#Executor	内存/Executor	任务并行度	节点数
35 分钟	75%	4	8	79	20 GB	79	10
38 分钟	100%	4	63	10	200 GB	150	10
43 分钟	50%	4	63	10	200 GB	150	10
44 分钟	100%	4	8	79	20 GB	158	10
34 分钟	75%	8	16	39	50 GB	39	10
45 分钟	100%	4	16	39	50 GB	150	10
46 分钟	100%	8	16	39	50 GB	78	10

极客时间

小结

只有理解 Spark 性能调优的本质，形成系统化的方法论，我们才能避免投入时间与产出不成正比的窘境。

性能调优的本质，我总结为 4 点：

1. 性能调优不是一锤子买卖，补齐一个短板，其他板子可能会成为新的短板。因此，它是一个动态、持续不断的过程；
2. 性能调优的手段和方法是否高效，取决于它针对的是木桶的长板还是瓶颈。针对瓶颈，事半功倍；针对长板，事倍功半；
3. 性能调优的方法和技巧，没有一定之规，也不是一成不变，随着木桶短板的此消彼长需要相应地动态切换；
4. 性能调优的过程收敛于一种所有木板齐平、没有瓶颈的状态。

系统化的性能调优方法论，我归纳为 4 条：

1. 通过不同的途径如专家经验或运行时诊断来定位性能瓶颈；

2. 从不同场景（典型场景）、不同视角（硬件资源）出发，综合运用不同层面（应用代码、Spark 配置项）的调优手段和方法；
3. 随着性能瓶颈的此消彼长，动态灵活地在不同层面之间切换调优方法；
4. 让性能调优的过程收敛于不同硬件资源在运行时达到一种平衡、无瓶颈的状态。

每日一练

1. 你还遇到过哪些“照本宣科”的调优手段？
2. 你认为，对于性能调优的收敛状态，即硬件资源彼此之间平衡、无瓶颈的状态，需要量化吗？如何量化呢？

关于性能调优，你还有哪些看法？欢迎在评论区留言，我们下节课见！

提建议

上一篇 01 | 性能调优的必要性：Spark本身就很快，为啥还需要我调优?

下一篇 03 | RDD：为什么你必须要理解弹性分布式数据集?

精选留言 (8)

写留言

October
2021-03-17

老师讲到可以通过运行时诊断来定位系统瓶颈，可借助于spark ui以及系统级监控工具，但是我依然不清楚怎样查看spark ui的各个指标，怎样查看每个应用程序的各种硬件负载，不知道老师后边的课程有没有相关内容

作者回复: 好问题，不过专栏最开始的计划没有包含这部分，原因是这部分其实网上都能查到，不过如果后续大家对这块疑问较多，到时候咱们也可以加餐，专门说说监控的事儿。

10 8

seed
2021-03-17

1. 还遇到的调优手段：直接从网上copy过来一些参数，在没有理解真正的原理的情况下，先怼上去，跑一下，任务时间缩短了就算调优了
2. 对于性能调优的收敛状态，需要量化；如何量化这些指标?
其实就是从我们需要调优的点出发；文中提到：从硬件的角度来看，计算负载划分为计算密集型、内存密集型和 IO 密集型。...

展开

作者回复: 说得非常好!

1的场景非常常见，很多时候都是知其然，而不知其所以然，稀里糊涂就算完事了，交差就行。

2的回答满分💯，无可挑剔。

2 6

裘元飞
2021-03-18

希望老师可以稍微讲一讲例如spark UI等工具如何监控指标等，主要很多时候都不知道有这些工具的存在。

作者回复: 可以的，后面找机会讲一讲，到时候看大家需要，可以开个直播当面讲，比较直观，图文讲UI很多细节都说不到。

3

Shockang
2021-03-17

王家林，段智华，夏阳编著的《Spark大数据商业实战三部曲》里面提到——大数据性能调优的本质是什么?答案是基于硬件的调优,即基于CPU、Memory、I/O(Disk/Network)基础上构建算法和性能调优! 无论是Hadoop,还是Spark,还是其他技术,都无法逃脱。 老师也在文章中提到——性能调优的最终目的，是在所有参与计算的硬件资源之间寻求协同与平衡，让硬件资源达到一种平衡、无瓶颈的状态。 我认为有异曲同工之妙！

展开

作者回复: 没错，性能调优的本质，其实就是用软件驱动硬件，让硬件在运行时能够高效协同，充分压榨每一类硬件资源的“剩余价值”。最早在IBM那会，可能是2011年吧，我们做IBM智能分析系统一体机，就是用软硬搭配的方式做数仓。那会对于硬件资源的协同，我们甚至有量化的比例，我记得是1:4:8，1cpu：4g内存：8块SAS盘。那会很早了，还用RAID阵列来提高I/O效率，当时网络用了最高配置，应该是万兆网我记得。虽然说场景不同，但是以硬件资源平衡为导向的思路，如出一辙。

2

2

L3nvy
2021-03-17

1. 不知道怎么分析性能瓶颈问题，按照自己的理解把业务逻辑Spark SQL重写了一遍，虽然时间下降了，但是引入了数据质量等问题。
2. 需要监控组件对机器的状态和系统metrics进行收集对比优化效果

展开

作者回复: 没错，1是典型的盲目调优，2的思路很赞～

2

Gnnn
2021-03-17

简单的增加并行度很多时候是没有太大效果的

展开

作者回复: 没错，并行度需要结合很多东西、尤其是cpu和内存方面的配置一起调整，才能起作用。

1

Fendora范东_

2021-04-02

磊哥说到了SQL functions指的是sparkSQL内置函数吧。
还提到了udf和sql functions的性能差异，那如果我用内置函数方式开发udf，理论上性能就是一样的?

作者回复: 一般来说，sql func会比udf强3%～5%，不等。但这只是他们本身的对比，实际在端到端的作业能提升多少，具体还有看场景。但能用sql func就尽量不用udf指定是没错～

joan

2021-03-23

老师，spark作业运行的网络性能怎么监控？目前我们只有shuffle读写量和速率相关，还有其他手段吗？谢谢！

展开

作者回复: 可以考虑结合ganglia做资源监控哈～ UI比较直观