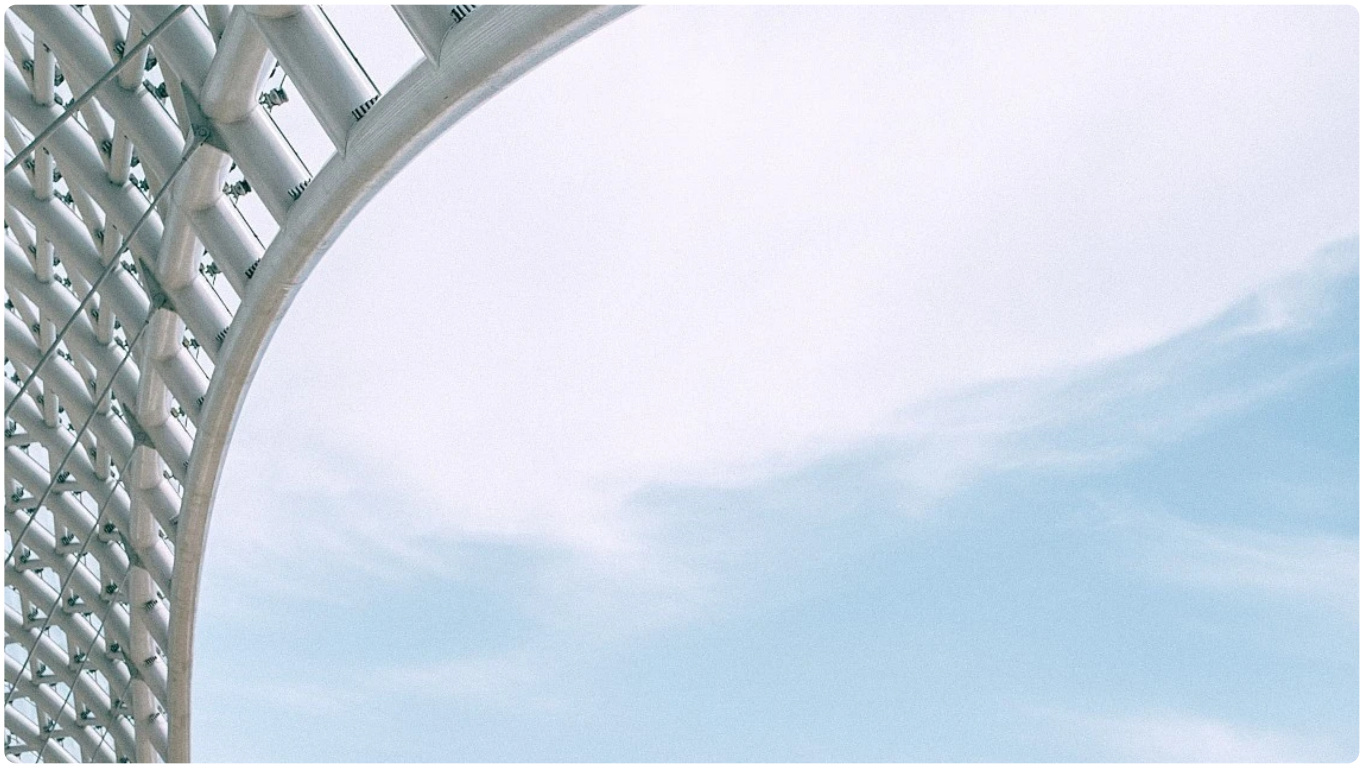
35 | Borg（二）：互不“信任”的调度系统

2022-01-10 徐文浩

《大数据经典论文解读》 课程介绍 >



讲述：徐文浩
时长 13:56 大小 12.77M

你好，我是徐文浩。

在上节课里，我们一起学习了 Borg 的整体架构。从架构层面来看，Borg 和其他的 Master-Slave 系统，其实都是类似的。其中比较大的一个挑战，是 Borg 需要管理万级别的机器。虽然 Borg 的 Master 集群，仍然是一个会选举出 master 的 Paxos 实现。但是除了 master 之外的其他副本，也需要去承担和 Borglet 通信的职责，而不仅仅是一个同步数据的副本。不过除此之外，在整体架构上，Borg 就没有太多的特殊之处了。

而对于 Borg 来说，真正的挑战还是在对于一个个 Task 的调度上。Borg 需要回答三个问题：

第一个问题是，当一个 Job 被发给 Master 的时候，我们究竟应该把它的 Tasks 调度到集群的哪一台机器上去？

第二个问题是，如果用户提交的 Job，Task 里实际消耗的资源和它申明的资源差异很大怎么办？

第三个问题是，在不同的 Task 之间互相竞争 CPU 资源的情况下，虽然看似 CPU 的利用率很高，会不会所有的资源都用在了上下文的切换上，而不是实际的 Task 的运算上？

那么今天这节课，就是要来回答这三个问题，理解了这三个问题，对于我们理解设计系统的工程实践会很有帮助。

我们设计的系统，不只能停留在自己的理论思考之中，还需要考虑实际用户会如何使用我们的系统，把“人”的因素一并考虑进来，才能设计出经得起时间考验的大型系统。

“贪心”的开发者和 Borg

每一个使用 Borg 的开发者，在向 Borg 提交任务的时候，都需要申明任务需要使用的资源。Borg 最多只会为你分配这些资源，如果你申请的资源太少，比如内存不足，导致你的程序运行不下去，那么这个就是你的问题了。

那么，作为开发者你会怎么做呢？要是我，那肯定是尽量多申请点资源给自己留一点余量。

大部分情况下，开发者并不会去仔细测试自己的程序到底会使用多少资源，很容易作出“拍脑袋”的决策。而且，一般来说，开发者都会偏向于高估自己所需要使用的资源，这样至少不会出现程序运行出问题的情况。但是，我们使用 Borg 的目的，就是尽量**让机器的使用率高一点**。每个开发者都给自己留点 Buffer，那我们集群的利用率怎么高得起来呢？

所以，面对贪心的都会多给自己申请一点资源的开发者，Borg 是通过这样两个方式，来提升机器的使用率。

第一个办法，是对资源进行“超卖”。

也就是我明明只有 64GB 的内存，但是我允许同时有申明了 80GB 的任务在 Borg 里运行。当然，为了保障所有生产类型的任务一定能够正常运行，Borg 并不会对它们进行超卖。但是，对于非生产类型的任务，比如离线的数据批处理任务，超卖是没有问题的。大不了，其中有些任务在资源不足的时候，会被挂起，或者调度到其他机器上重新运行，任务完成的时间需要久一点而已。

第二个办法，则是对资源进行动态的“回收”。

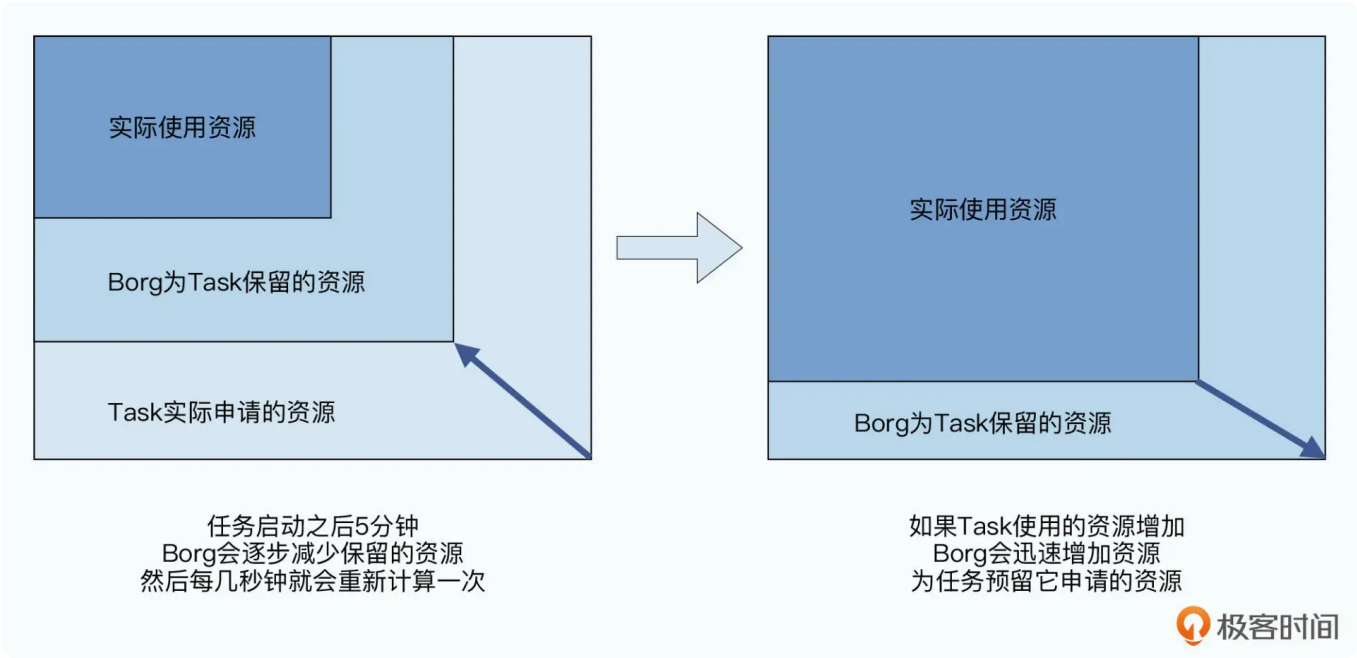
虽然对于非生产的 Job，我们已经采取了超卖的手段。不过，我们所有的生产的 Task，肯定也没有利用满它们所申请的资源。所以，Borg 实际不会为这些 Task 始终预留这么多资源。

Borg 会在 Task 开始的时候，先为它分配它所申请的所有资源。然后，在 Task 成功启动 5 分钟之后，它会慢慢减少给 Task 分配的资源，直到最后变成 Task 当前实际使用的资源，以及 Borg 预留的一些 Buffer。

当然，Task 使用的资源可能是动态变化的。比如一个服务是用来处理图片的，平时都是处理的小图片，内存使用很小，忽然来了一张大图片，那么它使用的内存一下子需要大增。这个时候，Borg 会迅速把 Task 分配的资源，增加到它所申请的资源数量。也就是说，无论是生产类型的 Task，还是非生产类型的 Task，Borg 实际上都会动态调整分配给它的资源。

我们把开发者申请的资源被称之为**限制资源（Resource Limit）**，而实际 Borg 动态分配给它的则是**保留资源（Resource Reservation）**，这两者的差值就是我们的**回收资源（Resource Reclamation）**。这部分回收资源，就是我们可以再利用起来的了。

不过需要注意，这部分资源，Borg 只会分配给非生产类型的任务。因为，这部分资源的使用是没有保障的，随时可能因为被回收了资源的生产类型 Task，忽然需要资源，被动态地抢回去。如果我们把这部分资源分配给其他生产类型的 Task，那么就会面临两个生产类型的 Task 抢占资源的问题。而我们在上一讲就说过，Borg 是不允许生产类型的 Task 相互抢占的。



从上图中我们可以看到，面对“贪心”的开发者，Borg 采用了所有申请都照章收取的方式。但是在实际分配的过程中，它其实是根据实际情况回收资源，并对资源进行超卖的方式，来提高集群的利用率。在论文的 5.5 部分和图 10 中显示，在 Google 里，20% 的任务是运行在回收而来的资源里的，这就给我们节约了 20% 的机器和相应的电费。可见，这两个策略是相当有效的。

平均分配还是多留点闲人

好了，我们已经了解了 Borg 是怎么通过超卖和回收资源的方式，来提高机器的利用率了。不过，当一个 Task 过来的时候，我们的调度器（Scheduler），应该把它具体分配到这 1 万台服务器里的哪一台服务器上呢？

首先，**Borg 里的调度器和 Master 是分离的**。调度器是异步从 Master 写入的队列里，去扫描有哪些 Task，然后再进行分配的。这个扫描过程中，调度器会先调度高优先级的，再调度低优先级的。而为了保障公平，在同一个优先级里，Borg 会采用**轮询**的方式。

然后，调度的过程分成两步。第一步，我们当然要先保障这个 Task 能正常运行，所以只能寻找哪些服务器能够满足被调度的 Task 的资源需求。这个，被称之为**可行性检查**（feasible checking）。然后是第二步，Borg 会对这个 Task 分配到每一台机器上，去打个分，**根据打分高低来选择一台服务器**。而当资源不足的时候，高优先级的 Task 会抢占掉已经在运行的，低优先级的 Task。

这个时候你可能要问了，我们随便选一台有足够资源的服务器不就行了么，打这么个分数有什么用呢？

那我们就来看看，如果我们随便写个简单的策略，可能会遇到什么情况。

Borg 一开始采用了这样一种策略，它是把每台服务器当前已经分配好的资源计算一个成本（Cost）。然后呢，模拟把 Task 分配到不同的服务器，再计算一个成本。两个成本之间差值最小的，也就是 Task 分配之后成本变化最小的，就是 Borg 选择的分配方案。

实践下来呢，最终 Task 分配会变成“**平均分配**”。也就是，每台服务器的负载都会尽量差不多，类似于一个“负载均衡”的方案。但这个问题是，如果我们有一个新的高优先级任务，需要一整台服务器的资源，比如 64 个 CPU、128G 内存。这个时候，我们集群里可能找不出任何一台服务器能有这样的资源。

那么，我们可不可以尽量把现在正在运行的服务器“塞塞满”，尽量多留出一些空闲的服务器呢？这样，如果有一个需要大量资源的 Task，我们就比较容易找到这样完全空闲的服务器了。

不过，这个策略也有对应的坏处，那就是一旦某个任务需要的资源忽然变多了，就会**抢占**掉同一台机器里面的非生产 Task 的资源。这个时候，那些任务要么得要挂起，要么就得另找机器重新运行了。而且，如果我们的 100 台服务器，10 台是满负载的，90 台是空闲的。一旦我们一台满负载的服务器出现硬件故障，那么对应就有 10% 的任务会受到影响，需要重新调度到其他服务器去。

即使我们的各种应用都做好了容错处理，**容错恢复也需要时间**。这个策略也会放大故障为我们业务带来的风险。

所以，最终 Google 选用了一个混合的模型来打分，也就是**采取尽量减少被“搁浅（stranded）”的资源数量**。所谓被“搁浅”，也就是说，当某个任务 100% 占用了自己所申明的资源之后，这个机器上会不能被使用的资源。

事实上，打分需要考虑的因素是很多的，资源分配只是其中的一部分，Borg 里至少还会考虑到下面这些因素：

比如我们应该**尽量挑，对应的机器不需要重新去下载对应 Task 的程序包的机器**。也就是服务器本地已经有了容器镜像的机器。这样，我们就不需要去重新下载 Task 的程序包，可以减少启动所需要的时间。

再比如，我们应该**尽量不要去抢占正在运行 Task 的资源**，如果不得不抢占的话，也尽量挑选抢占得少的机器。因为不是所有被抢占的任务，都是可以先挂起，然后恢复之后继续执行的。很多任务，一旦被抢占了，就执行失败，下次运行又要重新来过。可能看似我们集群的利用率很高，但是都是在做无效劳动。

还有，对于分布式服务的很多个 Task，我们尽量应该让它们**分布到不同的物理位置**，比如不同的机架、交换机等等。这样主要是为了避免，当出现电力故障、网络故障的时候，导致这样的 Job 出现单点故障。

最后，我们应该**让高优先级和低优先级的任务尽量混合部署**。而不是把所有高优先级的 Task 部署在一起，这样我们在负载峰值的时候，就很难让高优先级的任务，去增加自己需要使用的资源。

Borg的调度器核心策略

事实上，在 Borg 之后，Google 开源的 Kubernetes 中，你还可以根据自己的业务特性，去自定义自己的调度器，来满足你自己的需求。

专心致志和多线程工作的差别

可以看到，Borg 对于我们不同类型的任务，其实会采用**混合部署**的模式。也就是一台服务器上，既会有在线的、高优先级、提供服务的生产任务，比如一个 Nginx 反向代理服务器、Kafka 消息队列；也会有离线的、低优先级的批处理任务，比如一个数据分析师提交

的一次性分析任务。而且，我们通常会超卖我们的服务器，我们分配给各个 Task 的资源，也不是完全的 1 个 CPU，而可以是 0.1 个 CPU 这样的细粒度。

这就带来了一个问题，那就是不同的任务之间，其实在竞争相同的 CPU 资源，也会有大量的 CPU 上下文的切换。那么，这个情况是不是会导致，我们集群的利用率看似很高，但是实际上都是浪费在这些上下文切换里呢？这个对我们实际的集群影响会有多大？

说实话，这个问题没有办法得出理论上或者分析上的答案。因为每个 Borg 集群上，运行的任务都不一样，同一个 Borg 集群里，不同时间的负载也不一样，而且同一个集群里，不同的硬件可能也不一样。

不过，Borg 里还是用了一种“从底层解决问题”的方案，那就是直接去测算所有在运行的 Task 的 **CPI**（Cycles Per Instruction），也就是实际运行的 Task 里，每条 CPU 指令所需要的 CPU 时钟周期。一方面，这个参数隔绝了硬件差异的影响，另一方面，它也直接体现了，我们的 Tasks 运行的实际速度。CPI 大，意味着同样的 Task 需要花更多的时间。

那么，**混合部署之后，Borg 上运行任务实际的性能，有没有受到影响呢？**

答案是有影响，但是很小，相比于 CPU 利用率的提升，是完全划得来的。在 Borg 的性能评估里，向服务器添加 Task，会让原先的那些 Task 的 CPI 增加 0.3%，可以说忽略不计了。而整体上，CPU 利用率提升 10%，CPI 则只增加了 2%。这也就是说，虽然我们混合部署了更多的任务，的确有了更多的“额外开销（Overhead）”，但是大部分 CPU 资源，还是用在了新任务的计算上了。

从这个测算 CPI 的评估策略里你可以看到，如果你对计算机底层原理比较熟悉，你就更容易脱离开纷纷扰扰的上层应用系统，直接从底层原理里找到想要的答案。

小结

好了，到这里，对于 Borg 论文的讲解也就完结了。

由于 Borg 系统的用户是千千万万个内部的工程师，所以大有可能每个人都会申请过多的资源。Borg 的解决方案就是会尝试定时回收资源，不断从用户申请使用资源逼近到实际任务真正需要的资源上。

而在分配任务的时候，Borg 采取了一个折中的办法，它既需要考虑尽量让所有服务器的资源使用是平均分配的，使得后续的任务资源的动态扩展有可行性，但又要避免碎片资源太多，任何一台服务器都没有足够的资源，去分配之后出现需要大型资源的任务。而实际要考虑的任务分配因素，还要考虑程序包是否已经下载、同一个 Job 的不同任务是否分配到不同的物理位置，来确保高可用、尽量混合部署等一系列的因素。

最后，Google 还通过 CPI 这个最底层的指标，衡量了混合部署大量应用程序之后的性能影响。我们可以看到，Borg 在混合部署了多种不同的应用之后，CPI 的增加只有 2% 而已。而 CPU 的利用率则大大增加了，可以说是非常划得来了。

并且，Borg 是一个非常面向实战的系统，而不只是局限于理论。这个不仅是 Borg 这个系统的特性，也是我们之前看过的大量大数据系统的共性。

当然，Borg 也并不是完美的，这也是为什么后续 Google 并不是简单地开源了 Borg，而是进一步推出了 Kubernetes。那么，Kubernetes 相较于 Borg，究竟做了哪些改进呢？而 Kubernetes 比起 Borg 是否也还有什么做不到的事情吗？那就请你不要忘了我们下一讲，关于 Kubernetes 的论文解读。

推荐阅读

如果你对 Borg 集群的调度算法感兴趣，那么 Borg 里引用的《An opportunity cost approach for job assignment in a scalable computing cluster》这篇论文，就很值得一读了。虽然里面的数学公式看起来会有些多，但是通过形式化的方式来思考和设计算法，而不是仅仅凭直觉和经验，会让你对设计出来的系统是否完善，更有信心。

思考题

作为 Kubernetes 的前身，大型集群管理系统，Borg 的论文其实不是一两讲就能全面覆盖的。我们在前面的课程里，也只是主要讲解了 Borg 的系统架构，以及对于资源调度的策略。那么，在读过论文原文之后，你觉得 Borg 里还有哪些值得称道的设计和思考呢？

欢迎在留言区说说你的看法，也欢迎你把今天的内容分享给更多的朋友。

赞 0 提建议

上一篇 34 | Borg（一）：当电力成为成本瓶颈

精选留言 (1)

写留言

bearlu
2022-01-10

老师，Task 使用的资源可能是动态变化的，是用什么技术动态实现？