大规模集群调度系统综述

UNIkeEN

2023 年 12 月 10 日

摘要

本文分类讨论了多种集群调度系统，总结了互联网企业大规模集群调度系统面临的挑战，并探索了其在/离线任务混合部署的解决方案，揭示出集群调度技术的发展趋势和业务应用重要性。

关键词 集群调度、混合部署

1 问题描述与难点

集群调度系统是互联网企业核心的 IaaS 基础设施，用于解决数据中心的资源管理和任务调度问题，提升资源利用率。近年来，随着各公司业务快速增长，其服务器规模和数据中心数量暴增，大规模的集群调度面临多种挑战，具体如下：

- **吞吐量下降**：如目前业界较常用的开源容器编排系统 Kubernetes（K8s）[1] 在集群规模达数千个节点时，由于调度算法的计算复杂性，其调度吞吐量下降，影响调度性能 [2]。
- **工作负载感知**：如 K8s 原生调度器缺乏工作负载感知能力，静态调度可能引发负载不均衡或资源碎片化现象。且其默认不考虑属于同一应用程序的 pod 之间的关系，也无法执行如在具有相同网络前缀或具有同质设备的特定节点运行任务等高级操作 [2, 3]。
- **异构资源与多样化需求**：不同业务的多样化工作负载有不同的拓扑偏好，需要在异构资源上调度。满足所有约束条件并保持高资源利用率、向集群调度器集成新的资源（如 AI 加速芯片）具有挑战性 [2, 3, 4]。

2 集群调度系统分类

以下整理了目前业界较先进的集群调度器，一共分为四类：

2.1 单体调度器

单体调度器由一个中心化的调度器调度整个集群。其使用复杂的调度算法结合集群全局信息，计算高质量的放置点，如谷歌的 Borg[5, 6] 与开源的 Kubernetes[1]。K8s 的原生调度器按顺序运行一系列算法，主要包含排序、过滤和打分三个阶段，过滤和打分阶段以“扩展点”的形式执行多种系统或用户提供的算法。

2.2 双层调度器

单体调度器执行算法复杂，节点较多时会过载而出现高延迟，且在服务具有不同需求的异构工作负载方面存在问题。双层调度器将资源调度和任务调度分离，通过中心协调器动态分配资源给不同的工作负载，而每个工作负载又可以执行各自不同的作业调度逻辑，部分解决了大规模集群吞吐率与多样化调度策略的问题。

以 Apache Mesos [7] 为例，其 Mesos Master 组件负责管理集群资源，并将资源的“报价”发送给各个框架。每个资源报价包含了可用的资源量（如 CPU、内存等）；而各个框架有自己的调度器（Framework Scheduler）。当框架接收到资源报价后，它们自行决定是否接受这些资源并运行任务。代表性的双层调度器还有 Yarn [8]。

2.3 共享状态调度器

双层调度器缺乏全局可视性，容易产生资源碎片化、次优调度的问题，无法实现高优先级应用的抢占。共享状态调度器通过半分布式的方式来解决两级调度器的局限性，共享状态下的每个调度器都拥有一份集群状态的副本，且调度器独立对集群状态副本进行更新。一旦本地的状态副本发生变化，整个集群的状态信息就会被更新 [9]，代表如微软的 Apollo [10]。

Apollo 为每个作业分配一个调度器来管理作业的生命周期，并根据资源监视器的全局状态信息进行基于估算的调度。当调度器同时做出相互竞争的决策时，Apollo 乐观地推迟任何纠正行动，因为大多数冲突都是无害的。

此类调度器的缺点是持续资源争抢会导致调度器性能下降，且状态副本更新有时延。

2.4 分布式/混合调度器

使用较为简单的调度算法以实现大吞吐量调度 [11]。但由于缺乏全局资源使用视角效果不佳。另有混合调度器结合了单体或共享状态的设计。

3 面向业务场景的解决方案

3.1 在/离线任务混合部署

调度系统的好坏也取决于实际的调度场景。以业内使用最广泛的 Yarn 和 Kubernetes 为例。在实际应用时 Yarn 专注于离线批处理短任务，更倾向于低成本、高速度；Kubernetes 专注于在线长时间运行的服务，倾向于保障质量。

大型互联网企业通常拥有较多不同类型的业务，其大致可分为两种类型——在线业务，和用户存在交互的业务（如搜索、推荐、支付等业务），其特征是对时延十分敏感，负载存在峰谷波动，对错误容忍度较低；离线任务如机器学习的训练、视频转码、MapReduce 作业等，其特征是对时延不敏感，允许失败重试。

传统的数据中心将在、离线服务分开调度，维护、管理复杂、成本高昂。在线任务为保证服务性能和稳定性，通常按波峰申请资源，导致资源浪费（如网易广告推荐服务 CPU 利用率闲时仅 20%[12]）。而如今的互联网企业大多使用混合部署的形式对资源统一管理调度，将在线服务的空闲资源用于离线任务，提升资源利用率。

统一的调度系统会面临前述的研究难点。以下对部分企业的实际方案进行总结：

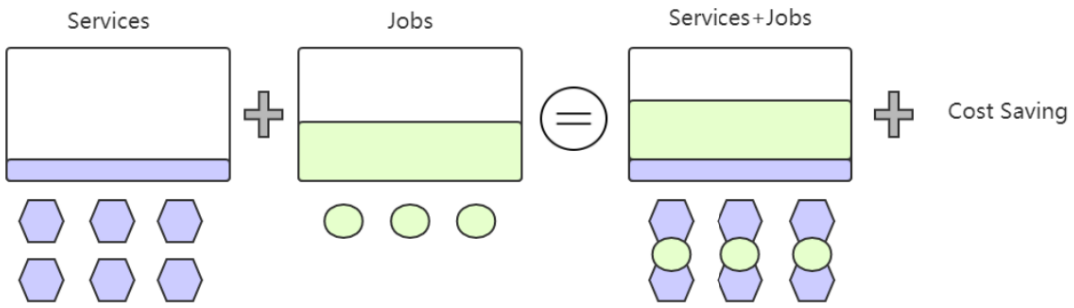

Fig. 1. 混合部署示意图

3.2 动态调度

原生 Kubernetes 根据容器资源请求实行静态调度而不考虑实际负载，在线任务节点闲时负载低但无法调度新的 pod。针对此问题，各企业都引入了动态资源视图以进行工作负载感知。

网易 [12]、美团 [4] 和百度 [13] 都基于 Kubernetes 的原生实现开发扩展插件，将在线、离线任务分开调度并各自提供独立的资源视图。对于在线任务，其看到的可用资源仍为整机资源，在离线任务之前进行原有策略的静态分配；进行离线任务分配时，动态计算剩余资源进行分配——剩余资源包含空闲资源和已请求、未使用的可回收资源。如网易将可回收资源单独命名为 colocation/cpu 和 colocation/memory（分别对应原生的空闲 cpu 和内存资源），在调度 pod 前由各节点的 monitor-agent 进行汇报 [12]。

同时，在 Kubernetes 的 QoS（服务质量）模型中，BestEffort 不设置也不占用资源请求量，网易 [12] 和百度 [13] 都将离线任务复用 BestEffort 模型，同时设置离线任务的最高资源占用量。对离线任务复用 BestEffort 模型的好处是，原生 Kubernetes 会在单机负载过高时自动驱逐此类 pod，避免影响在线任务的稳定性与服务质量。

而面对更复杂的场景，小红书 [14] 和字节 [2] 均使用 Kubernetes 和 Yarn 分别部署管理在、离线任务。Yarn 的核心组件是 ResourceManager（RM）和 NodeManger（NM），前者在管控侧负责接收任务以及资源调度，后者管理任务的生命周期 [8]。小红书使用 Koordinator[15, 14] 将 RM 独立部署，将 NM 和 Yarn 任务分别作为 pod 在 K8s 的节点上部署，均使用 BestEffort 策略。

字节则在近期设计了全局共享状态调度器 Gödel[2]。

3.3 提升吞吐量

离线任务相比于长期常驻运行的在线任务，运行时间段、任务多。原生 Kubernetes 调度器的调度性能不足以支撑，测试显示其在集群规模超过 500 个节点时调度吞吐量开始下降 [2]。百度自行开发了高性能离线调度器 [13] 提升吞吐量。

Yarn 在批处理任务（离线）的调度吞吐量显著高于 [10]，但其无法原生调度在线服务甚至在同一资源池调度异构负载。小红书 [14] 打通了 Yarn 调度器和 Kubernetes 调度器的资源视图，在 Yarn 和 Kubernetes 间动态同步实际资源使用状态。继续使用 Yarn 的调度器调度离线 Spark 算法类任务，以利用 Yarn 的高吞吐量优势。字节在前期也使用此类并行调度方案 [2]，并实现了协调器持续监控两侧资源供应并移动资源。但这一方案的问题在于移动资源时可能造成连锁依赖故障。

字节新提出的 Gödel[2] 替换原生 Kubernetes 调度器，提供一系列并行决策的调度器实例，重新定义了调度单元（包含 pod/离线子任务），在调度器级别填补了在线和离线作业之间的语义差距，Yarn RM 将分配周期决策也外包给 Gödel。

同时，Gödel[2] 的多个调度器实例通过共享状态（目前通过调用 API 服务器到后备存储，测试显示改用内存缓存可将性能进一步翻倍）进行合作。在集群中存在大量资源，并且调度器实例之间的冲突可以忽略不计时，每个调度器实例会通过乐观并发控制在分区之间选择最佳节点。否则切换到分片模式，其中每个调度器实例只能从其拥有的分区中查找可行的节点以降低冲突率，这两项策略同样提升了调度器的吞吐量。

3.4 异构资源与拓扑感知

调度器在放置 pods 时应遵守资源拓扑约束，一种无法调度的情况是节点资源量足够、但资源分布无法满足拓扑策略。最佳解决方式是在调度时就拥有每个节点上可用资源的拓扑信息，而原生 Kubernetes 实现在调度后的 kubelet admit 阶段才会发现此情况并使 pod 启动失败，效率较低。此外，机器学习业务的引进带来了更复杂的拓扑偏好（如密集预载在内存的预训练模型）与异构资源（GPU）。

字节在 Gödel[2] 中引入了 CustomNodeResource (CNR)。集群的每个活动服务器都创建一个 CNR 对象，储存节点的 NUMA 信息、带宽、GPU 信息，在节点代理创建或删除 pods 时同步更新。调度时通过扫描 CNR 对内存密集型、网络密集型、机器学习任务等分别做出更好的决策，避免调度后启动失败。

除了以上方面，学术界、企业还在在/离线任务资源隔离 [4, 12, 13]、减少重调度成本 [12, 13]、根据历史负载与机器学习方法的预测 [4, 13, 16] 等方面对调度器进行了改进，在此受限于篇幅暂不讨论。

4 总结

本文调研了一些业界领先的集群调度系统相关论文和企业文档，对集群调度系统的种类进行了归纳总结。同时总结了互联网企业在大规模集群调度、在/离线任务混合部署方面遇到的挑战和相应的解决措施。

优先队列、抢占、异构调度、负载均衡..... 从单点到分布式，企业大规模集群调度系统的许多思想和《现代操作系统》课程所学的单/多核调度有许多共通之处，却又在业务稳定性、服务质量、利用率、成本等方面面临更多的挑战，这大大激发了我的兴趣，拓展了我思考问题的全面性，也深刻体会到云计算基础设施对于企业的重要性。在课内的实验中，我有幸使用华为云实践使用了 Kubernetes 并学习了原生 Kubernetes 调度器的实现，这也加深了我对这次调研的理解。

感谢李超老师一学期的教导与付出。

参考文献

[1] “Kubernetes.” https://kubernetes.io/.

[2] W. Xiang, Y. Li, Y. Ren, F. Jiang, C. Xin, V. Gupta, C. Xiang, X. Song, M. Liu, B. Li, K. Shao, C. Xu, W. Shao, Y. Fu, W. Wang, C. Xu, W. Xu, C. Lin, R. Shi, and Y. Liang, “Gödel: Unified large-scale resource management and scheduling at bytedance,” in *Proceedings of the 2023 ACM Symposium on Cloud Computing*, SoCC ’23, (New York, NY, USA), p. 308–323, Association for Computing Machinery, 2023.

[3] C. Carrión, “Kubernetes scheduling: Taxonomy, ongoing issues and challenges,” *ACM Comput. Surv.*, vol. 55, dec 2022.

[4] 谭霖, "美团集群调度系统的云原生实践." https://tech.meituan.com/2022/02/17/kubernetes-cloudnative-practices.html, 2022.

[5] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, "Borg: The next generation," in *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[6] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, (New York, NY, USA), Association for Computing Machinery, 2015.

[7] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for {Fine-Grained} resource sharing in the data center," in *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, 2011.

[8] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, (New York, NY, USA), Association for Computing Machinery, 2013.

[9] Y. Feng, Z. Liu, Y. Zhao, T. Jin, Y. Wu, Y. Zhang, J. Cheng, C. Li, and T. Guan, "Scaling large production clusters with partitioned synchronization," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 81–97, 2021.

[10] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, "Apollo: Scalable and coordinated scheduling for cloud-scale computing," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, (USA), p. 285–300, USENIX Association, 2014.

[11] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, (New York, NY, USA), p. 69–84, Association for Computing Machinery, 2013.

[12] 张晓龙, 李岚清, 陈林亮, "网易轻舟基于 k8s 的业务混部署实践." https://mp.weixin.qq.com/s/ouUOw3skhgYPomVVrVWeRQ, 2022.

[13] 百度, "深入理解百度在离线混部技术." https://zhuanlan.zhihu.com/p/453977108, 2022.

[14] 索增增, 宋泽辉, 张佐玮, "Koordinator 支持 k8s 与 yarn 混部, 小红书在离线混部实践分享." https://developer.aliyun.com/article/1371953?utm_content=g_1000383844, 2023.

[15] "koordinator documentation." https://koordinator.sh/docs, 2023.

[16] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, and C. Delimitrou, "Sinan: Ml-based and qos-aware resource management for cloud microservices," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, (New York, NY, USA), p. 167–181, Association for Computing Machinery, 2021.