1 国内外研究现状

综：

配置优化方面已有的相关工作主要集中在解决传统软件的性能问题，如Hadoop[1]、Spark[2]、Flink[3]等大数据框架提供了大量参数用以调节框架运行时性能。由于云原生环境下微服务应用具有组件化、依赖复杂等特点，这使得传统软件上相关工作很难被直接迁移到微服务应用中。虽然最近有部分工作开始关注于微服务应用的配置优化问题，但他们主要对微服务应用的资源配置或软件参数进行了独立的考虑，未能对两类配置参数进行协同优化，造成了性能优化空间的损失。

1.1 传统软件配置优化现状

综：

经过多年的发展，大数据已从一个新兴技术领域逐渐转变为社会经济发展中各个领域的重要因素、资源、动力与理念。针对大数据生态中的各个组件，如Hadoop、Spark、Flink 等，研究人员提出了一系列的方法来优化其性能。同时随着数据量的增加对数据库软件的性能优化也成为了一个重要的研究方向。本节分别对这两方面的相关工作进行介绍。

Mathiya[4]等通过实验证明，相较于 Hadoop 默认的参数配置，良好的自定义参数能够提高集群资源利用率，进而提升系统的性能，证明了对大数据软件进行配置优化具有重要意义。Herdotu 等[5]提出了 Starfish，通过建立精确的成本模型来模拟 MapReduce 的执行过程，并使用一个小型模拟器组件模拟任务调度决策，可以让用户在不熟悉 Hadoop 的内在原理的情况下，也能够对该系统进行优化。该作者还提出了 Elastisizer[6]将作业层面的软件参数与资源配置组合在一起，扩展了 Starfish。通过在小型集群和小型数据集上多次执行 MapReduce 作业，建立了回归树模型对运行成本开销进行预测。并且 Elastisizer 使用递归随机搜索来同时确定最佳集群大小和作业参数配置，以便在请求时间或成本预算内执行完 MapReduce 作业。Liao 等[7]提出了一个基于搜索的 Hadoop 调优系统Gunther，它将配置优化视为一个黑箱优化问题，使用遗传算法搜索参数配置，可以在不到 30 次的迭代下找到接近最优的参数配置。此外 Gunther 忽略了对性能影响不大的参数以减少搜索时间。

上述工作集中于 Hadoop 框架，除此之外，研究人员还对大数据生态下的其它软件配置优化问题进行了深入研究。Petridis 等[8]提出了一种基于少量实验来调整 Spark 框架参数的试错方法。具体来说，基于文档和过去的执行经验，

作者首先选择了 12 个重要的参数，然后在不同的基准应用下测试这些参数对性能的影响。基于测试结果，作者以方块图的形式开发了一种方法，其中包含 7 个对性能最具影响力的参数，然后通过从方块图中获取不同的参数配置，执行 Spark 负载进行配置优化。Yu 等[9]为内存数据处理框架（如 Spark）引入了一种参数自动调整方法，作者考虑到了数据量大小和高维参数方面的问题，他们首先提出了一个分层预测模型，该模型由许多分层排列的子模型组成，中心思想是建立几个较简单的模型，而不是单一的复杂模型。接下来，作者采用遗传算法（Genetic Algorithm，GA）来寻找最佳参数配置。Li 等[10]在其研究中提出使用生成对抗网络来优化 Spark 的配置参数，它可以通过使用较少的训练数据来构建性能预测模型，同时不会降低模型的准确性。除此之外，还使用了优化的遗传算法来搜索参数空间，以获得最佳的配置方案，在五个典型的工作负载上，搜索到的参数配置相比默认配置，Spark 性能均得到了提高。此外，Fekry 等[11]提出了 Tuneful 配置优化框架，可对数据处理框架 Spark 进行配置优化。Tuneful 对敏感度分析（Sensitivity Analysis，SA）方法进行了改进，采用多轮 SA 逐步选择关键参数，以提高优化效率。针对 Flink 框架，Guo 等[12]提出了一种引导式机器学习的方法来自动调整 Flink 的配置参数，作者先使用生成对抗网络生成一些样本数据为 Flink 的配置与性能之间建立模型，然后用引导式机器学习算法为 Flink 集群寻找最优配置，与其它基于机器学习的大数据软件自动配置优化方法相比，引导式机器学习的方法可以大大减少训练数据收集和最佳配置搜索所需的时间。Q-Flink[22]是一种适用于混布工作负载下的 Flink 共享资源控制策略，它监测共享资源在混布工作负载中相互竞争的情况，考虑不同工作负载对服务质量(Quality of Service，QoS)的不同要求，然后对共享资源进行分配，以减少作业服务质量的违规率。

随着大数据、云计算、互联网和移动互联网的发展，数据量呈指数级增长。在研究人员对大数据生态软件配置优化问题研究的同时，传统的关系型数据库也难以满足当今海量数据的存储和处理需求。MongoDB、Cassandra 等新型数据库在这个背景下逐渐变得流行起来。它们有着高性能和高可扩展性的特点，能够更好地处理大规模和高复杂度的数据。同时云计算的发展也使得数据库可以更加灵活地部署和扩展，更好地支持数据的分布式存储和处理。正确选择数据库软件的配置参数对提高性能和降低成本至关重要，因此数据库软件的配置优化问题也得到了研究人员的广泛关注。Van 等[13]等设计了一个自动调优工具 OtterTune，用于优化数据库管理系统的配置参数。OtterTune 利用历史经验，采用监督和非监督机器学习方法的组合，实现了选择关键参数、建立工作负载映射和生成推荐参数设置的目标。它采用 Lasso 算法选择关键参数，但每次调用需要大量的时间

和内存开销，以处理庞大的历史数据。该工具在 MySQL、Postgres 和 Actian Vector 三个数据库管理系统上进行了实验，虽然每个试验的测量时间很短，只有 5 分钟，但是仅仅为了收集初始数据 OtterTune 就花费了三个多月的时间。CGPTuner[14]使用贝叶斯优化对数据库管理系统进行调优，该系统并不需要收集大量初始数据进行引导，并且考虑了 Java 虚拟机、操作系统、物理机等层次的配置参数。在 Cassandra 和 MongoDB 上已经得到了很好的应用。Kanellis 等[15]针对数据库系统，使用分类回归树算法（Classification And Regression Tree，CART）选择关键参数，并且在构建随机森林的过程中可以捕获参数之间非线性的关系。作者通过实验证明只有少数几个关键参数（约 5 个）会对系统性能产生较大的影响，并且在不同负载下，关键参数是基本保持一致的，然而关键参数保持一致这个结论在微服务场景下是不成立的。

述：

上述相关研究局限于大数据软件和数据库等传统软件的配置优化问题，然而现今软件架构已经改变，微服务架构为越来越多公司接受和采用，尽管基于传统软件的配置优化方面的研究具有重要的价值，但是它们只考虑单个软件栈的参数调整，并未考虑多个软件、不同软件的参数相互影响等存在于微服务应用中的特点，因此传统软件上的配置优化工作难以直接应用到微服务中，需要进一步扩展配置优化相关工作研究的范围，以全面评估和提高微服务应用的性能。

1.2 微服务配置优化现状

综：

由于微服务应用包含多个服务、服务之间依赖复杂且参数之间存在相互影响，因此传统软件上的相关工作很难直接应用到微服务中。目前研究人员开始针对微服务场景下的配置优化问题进行研究，该场景下已有的工作主要分为两类方案：第一类通过优化 CPU、内存等资源配置以提高微服务应用性能；第二类通过优化微服务应用所部属的 Nginx、Redis 以及 MongoDB 等软件自身配置参数以提高微服务应用性能。下面将分别介绍这两类方案。

在资源配置优化方面，Reiss 等[16]通过研究谷歌公布的集群使用信息数据集[27]，指出集群中资源类型和使用方式的异质性，这种异质性会降低传统资源调度技术的有效性。Reiss 还指出集群中工作负载是高度动态的，由许多短期工作和少量具有稳定资源利用率的长期运行工作组成。总的来说，作者通过对谷歌公布的数据集进行研究，证明了在云计算环境下对资源配置进行优化的必要性。在此基础上 Jyothi 等[17]为了解决云计算服务端追求高资源利用率和客户端追求应用高性能两者之间的矛盾，先对用户服务等级目标（Service-level Objective，

SLO）进行了合理的定义，然后分析客户端作业之间的依赖关系，并利用历史信息进行建模，使用周期性预定（Recurring Reservation）的思想对服务端资源配置进行管理，在提高系统资源利用率的情况下降低了 SLO 违规的次数。Gias 等[18]设计了一个微服务应用系统资源自动扩缩控制器，对服务副本数和 CPU 资源用量进行调整，并且同时考虑了水平扩缩[19]和垂直扩缩[20]对微服务应用性能的影响。该控制器使用分层排队网络（Layered Queueing Network，LQN）模型对微服务应用性能、服务副本数、CPU 资源用量三者进行建模。最后使用遗传算法对最优参数配置进行搜索。FIRM[21]也对微服务应用的资源配置进行了研究，它是一个针对微服务应用资源管理框架，可以用于缓解微服务之间由于资源竞争导致的 SLO 违规，框架使用支持向量机（Support Vector Machine，SVM）检测服务性能异常，异常发生时，通过添加更多的 Pod[22]或 CPU、内存等系统资源，对相应的服务进行扩容。FIRM 的缺陷主要有两点，第一只有在性能异常发生后才会进行系统资源自动扩缩，并且大规模的自动缩放可能需要几分钟的冷启动，或者至少几秒钟的热启动，因此需要较长的响应时间，有可能使服务异常持续一段较长的时间。第二由于 FIRM 通过系统异常进行触发，当分配给微服务的资源超过实际所需用量时，它无法对系统资源进行回收，造成资源浪费。对于资源无法自动缩放的问题，Autopilot[23]基于滑动窗口算法设计的资源配置推荐器可以在服务转向平稳运行或者负载峰值消退后，回收之前过多分配的系统资源，一定程度上解决了 FIRM 无法对资源进行回收的问题，提高了资源利用率。Zhang 等[24]考虑到微服务应用中服务分层的特点，提出微服务应用资源配置管理模型 Sinan，它可以在线运行，根据服务的运行状态和端到端的 QoS 目标，动态地调整各层服务的系统资源用量。Sinan 首先使用空间探索算法来检查可能的资源配置搜索空间，通过收集的数据训练两个模型：卷积神经网络（Convolutional Neural Network，CNN）模型，用于微服务应用短期性能预测；提升树（Boosting Tree）模型，用于评估微服务应用长期性能演变。两个模型的结合使 Sinan 既能检查资源配置的近期效果，又能考虑到系统中工作负载的相似性，优化精度比单一模型更加准确。

上述研究主要集中在微服务应用 CPU、内存等资源配置的优化方面，另外微服务应用中所部署的软件也存在大量可供调节的参数，通过对软件参数调整也可以优化微服务应用的性能，目前通过调整微服务部署的软件参数对应用进行配置优化的相关研究较少，下面对该方案已有的研究进行介绍。Mahgoub 等[25]提出了 OPTIMUSCLOUD 系统针对云环境下的数据库软件进行配置优化，通过建模的方式对最优配置进行搜索。Wang 等[26]扩展了开源调优工具 KeenTune[27]，对企业级云原生应用软件和服务进行自动化配置优化，在 MySQL、

OceanBase、Nginx、Ingress-Nginx 等软件和服务上取得了不错的效果。然而这些工作局限于调整单个微服务应用中存在的软件，未对微服务全链路进行考虑。Somashekar 等[28,29]对微服务应用的配置优化工作进行了相对更加全面的考虑，通过协同调优作为服务部署的 Nginx、Memecached、MongoDB 等软件的配置参数来优化微服务应用性能。以上通过调整微服务应用软件参数，进行应用性能优化的工作局限性在于：未将软件参数与微服务资源配置进行协同考虑，而在峰值负载时调整资源配置能够为微服务应用带来更大的性能提升。

述：

综上所述，目前国内外配置优化问题的相关工作，主要关注解决传统软件性能问题，难以直接适用于微服务应用。目前已有的少量关于微服务应用配置优化的工作，分开考虑了资源配置和软件参数，没有将两者协同考虑，导致了微服务应用性能优化空间的损失。

参考文献

[1] Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system[A]. IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)[C]. Incline Village: IEEE, 2010: 1-10.

[2] Zaharia M, Chowdhury M, Franklin MJ, et al. Spark: Cluster computing with working sets[A]. 2nd USENIX Conference on Hot Topics in Cloud Computing[C]. Boston: USENIX Association, 2010: 10-10.

[3] Carbone P, Katsifodimos A, Ewen S, et al. Apache flink: Stream and batch processing in a single engine[J]. The Bulletin of the Technical Committee on Data Engineering, 2015, 36(4): 28-38.

[4] Mathiya BJ, Desai VL. Apache hadoop yarn parameter configuration challenges and optimization[A]. 2015 International Conference on Soft-Computing and Networks Security (ICSNS)[C]. Coimbatore: IEEE, 2015: 1-6.

[5] Herodotou H, Lim H, Luo G, et al. Starfish: A Self-tuning System for Big Data Analytics[A]. 5th Biennial Conference on Innovative Data Systems Research(CIDR)[C]. Asilomar: Online Proceeding, 2011: 261-272.

[6] Herodotou H, Dong F, Babu S. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics[A]. 2nd ACM Symposium on Cloud Computing[C]. Cascais: ACM, 2011: 1-14.

[7] Liao G, Datta K, Willke TL. Gunther: Search-based auto-tuning of mapreduce[A]. Euro-Par 2013 Parallel Processing: 19th International Conference[C]. Aachen: Springer, 2013: 406-419.

[8] Petridis P, Gounaris A, Torres J. Spark parameter tuning via trial-and-error[A]. Advances in Big Data: 2nd INNS Conference on Big Data[C]. Thessaloniki: Springer, 2017: 226-237.

[9] Yu Z, Bei Z, Qian X. Datasize-aware high dimensional configurations auto-tuning of in-memory cluster computing[A]. 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'18)[C]. Williamsburg: ACM, 2018: 564-577.

[10] Li M, Liu Z, Shi X, et al. ATCS: Auto-tuning configurations of big data frameworks based on generative adversarial nets[J]. IEEE Access, 2020, 8: 50485-50496.

[11] Fekry A, Carata L, Pasquier T, et al. To tune or not to tune? in search of optimal configurations for data analytics[A]. Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining[C]. Virtual Event: ACM, 2020: 2494-2504.

[12] Guo Y, Shan H, Huang S, et al. GML: Efficiently Auto-Tuning Flink's Configurations Via Guided Machine Learning[J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 32(12): 2921-2935.

[13] HoseinyFarahabady MR, Jannesari A, Taheri J, et al. Q-flink: A qos-aware controller for apache flink[A]. 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)[C]. Melbourne: IEEE, 2020: 629-638.

[14] Van Aken D, Pavlo A, Gordon GJ, et al. Automatic database management system tuning through large-scale machine learning[A]. 2017 ACM International Conference on Management of Data[C]. Chicago: ACM, 2017: 1009-1024.

[15] Cereda S, Valladares S, Cremonesi P, et al. Cgptuner: a contextual gaussian process bandit approach for the automatic tuning of it configurations under varying workload conditions[J]. VLDB Endowment, 2021, 14(8): 1401-1413.

[16] Kanellis K, Alagappan R, Venkataraman S. Too many knobs to tune? towards faster database tuning by pre-selecting important knobs[A]. Workshop on Hot Topics in Storage and File Systems[C]. Virtual Event: USENIX Association, 2020: 1-1.

[17] Reiss C, Tumanov A, Ganger GR, et al. Heterogeneity and dynamicity of clouds at scale: Google trace analysis[A]. third ACM symposium on cloud computing(SOCC)[C]. San Jose: ACM, 2012: 1-13.

[18] Wilkes J, Reiss C. Details of the ClusterData-2011-1 trace 2011[EB/OL], https://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1, 2011-01-01/2023-01-02.

[19] Jyothi SA, Curino C, Menache I, et al. Morpheus: Towards Automated SLOs for Enterprise Clusters[A]. 12th USENIX Symposium on Operating Systems Design and Implementation(OSDI)[C]. Savannah: USENIX Association, 2016: 117-134.

[20] Gias AU, Casale G, Woodside M. ATOM: Model-driven autoscaling for microservices[A]. 39th International Conference on Distributed Computing Systems (ICDCS)[C]. Dallas: IEEE, 2019: 1994-2004.

[21] Kubernetes Documention. How does a Horizontal Pod Autoscaler work[EB/OL],

https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/#how-does-a-horizontalpodautoscaler-work, 2022-11-26/2023-01-02.

[22] Google Kubernetes Engine Documentation. How vertical Pod autoscaling works[EB/OL], https://cloud.google.com/kubernetes-engine/docs/concepts/verticalpodautoscaler, 2022-12-27/2023-01-02.

[23] Qiu H, Banerjee SS, Jha S, et al. FIRM: An intelligent fine-grained resource management framework for slo-oriented microservices[A]. 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)[C]. Virtual Event: USENIX Association, 2020: 805-825.

[24] Kubernetes Documention. How Pods manage multiple containers[EB/OL], https://kubernetes.io/docs/concepts/workloads/pods/#what-is-a-pod, 2022-12-15/2023-01-02.

[25] Rzadca K, Findeisen P, Swiderski J, et al. Autopilot: workload autoscaling at Google[A]. Fifteenth European Conference on Computer Systems(EuroSys)[C]. Heraklion: ACM, 2020: 1-16.

[26] Zhang Y, Hua W, Zhou Z, et al. Sinan: ML-based and QoS-aware resource management for cloud microservices[A]. 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems[C]. Virtual Event: ACM, 2021: 167-181.

[27] Mahgoub A, Medoff A, Kumar R, et al. OPTIMUSCLOUD: Heterogeneous configuration optimization for distributed databases in the cloud[A]. 2020 USENIX Conference on Usenix Annual Technical Conference[C]. Virtual Event:USENIX Association, 2020: 189-204.

[28] Wang R, Wang Q, Hu Y, et al. Industry practice of configuration auto-tuning for cloud applications and services[A]. 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering[C]. Singapore: ACM, 2022: 1555-1565.

[29] OpenAnolis SIG. KeenTune[EB/OL], https://openanolis.cn/sig/keentune, 2022-11-01/2023-01-03.