Sinan：云微服务中基于机器学习和服务质量管理的资源管理模型

郭春浩 M202173682

摘　要　云应用程序正越来越多地从大型单片服务转向大量松散耦合的、专门的微服务。尽管微服务在促进开发、部署、模块化和隔离方面具有优势，但它们使资源管理复杂化，因为它们之间的依赖关系引入了背压效应和级联 QoS 违反。我们介绍了 Sinan，一个用于交互式云微服务的数据驱动集群管理器，具有在线和 qos 感知。Sinan 利用一组可伸缩的和经过验证的机器学习模型来确定微服务之间的依赖关系的性能影响，并以保留端到端尾部延迟目标的方式为每个层分配适当的资源。我们评估了专用本地集群上的 Sinan 和谷歌计算引擎(GCE)上的大规模部署，跨使用微服务构建的代表性端到端应用程序，如社交网络和酒店预订网站。我们表明，Sinan 总是满足 QoS，同时也保持了较高的集群利用率，而之前的工作会导致不可预测的性能或牺牲重新。

关键词　云计算、数据中心、服务质量、尾延迟、微服务、集群管理、资源管理、资源分配、资源效率、系统的机器学习

Sinan: ML-Based and QoS-Aware Resource Management for Cloud Microservices

Chunhao Guo

Abstract Cloud applications are increasingly shifting from large monolithic services, to large numbers of loosely-coupled, specialized microser vices. Despite their advantages in terms of facilitating development,deployment, modularity, and isolation, microservices complicate resource management, as dependencies between them introduce backpressure effects and cascading QoS violations. We present Sinan, a data-driven cluster manager for interactive cloud microservices that is online and QoS-aware. Sinan lever ages a set of scalable and validated machine learning models to determine the performance impact of dependencies between mi croservices, and allocate appropriate resources per tier in a way that preserves the end-to-end tail latency target. We evaluate Sinan both on dedicated local clusters and large-scale deployments on Google Compute Engine (GCE) across representative end-to-end applications built with microservices, such as social networks and hotel reservation sites. We show that Sinan always meets QoS, while also maintaining cluster utilization high, in contrast to prior work which leads to unpredictable performance or sacrifices resource efficiency. Furthermore, the techniques in Sinan are explainable, meaning that cloud operators can yield insights from the ML models on how to better deploy and design their applications to reduce unpredictable performance.

Key words Cloud computing, datacenter, quality of service, tail latency, microservices, cluster management, resource management, resource allocation, resource efficiency, machine learning for systems

1、引言

近年来，云应用程序已经逐渐从单片服务转变为具有数百个单一用途和松散耦合的微服务的图形服务。这种转变正变得越来越普遍，因为大型云服务提供商，如亚马逊、推特，已经采用了这种应用程序模式。

尽管有几个优点，如模块化和灵活的开发和快速迭代，微服务也引入了新的系统挑战，特别是在资源管理，因为微服务依赖的复杂拓扑加剧了排队

———————————————

效应，并引入级联服务质量(QoS)违规，很难及时识别和纠正。当前的集群管理器是为单片应用程序或由一些流水线层组成的应用程序设计的，并且不足以捕捉微服务的复杂性。考虑到越来越多的生产云服务，如推特和亚马逊，现在被设计为微服务，解决它们的资源管理挑战是一个迫切需要。

我们采用数据驱动的方法来解决资源管理中引入的复杂性。类似的机器学习(ML)驱动的方法，在以前的工作中已经有效地解决了大规模系统的资源管理问题。不幸的是，这些系统并不直接适用于微服务，因为它们是为单片服务而设计的，因此没有考虑到微服务之间的依赖关系对端到端性能的影响。

我们提出了 Sinan，一个可扩展的和 qos 感知的资源交互式云微服务管理器。思南没有要求用户或云运营商推断微服务之间依赖关系的影响，而是利用一组经过验证的 ML 模型来自动确定每层资源分配对端到端性能的影响，并为每层分配适当的资源。

Sinan 首先使用一种有效的空间探索算法来检查可能的资源分配的空间，特别是关注引入 QoS 违规的角情况。这就产生了一个用于训练两个模型的训练数据集：一个用于详细的短期性能预测的卷积神经网络(CNN)模型，以及一个用于评估长期性能演化的增强树模型。这两种模型的结合使 Sinan 既可以检查资源分配的近期结果，又可以考虑系统在建立队列时的惯性，并具有比检查两个时间窗口的单一模型更高的精度。Sinan 在线运行，根据服务的运行时状态和端到端 QoS 目标动态调整每层资源。最后，Sinan 被实现为一个集中的资源管理器，具有对集群和应用程序状态的全局可见性，并具有跟踪每层性能和资源利用率的每个节点的资源代理。

我们使用两个端到端应用程序来评估 Sinan，该应用程序使用交互式微服务：一个社交网络和一个酒店预订网站。我们比较了 Sinan 与传统使用的经验方法，如自动缩放，以及之前基于队列分析的多层服务调度的研究，如 PowerChief。我们证明了 Sinan 在性能和资源效率方面都优于以前的工作，成功地满足了不同加载模式下的两个应用程序的 QoS。在更简单的酒店预订申请中，Sinan 平均节省了 25.9%，以及高达其他 qos 会议方法所使用的资源量的 46.0%。在更复杂的社交网络服务中，抽象应用程序的复杂性更为重要，Sinan 平均节省了 59.0%的资源，甚至高达 68.1%，基本上可以容纳每秒两倍的请求量，而不需要更多的资源。我们还通过在谷歌计算引擎(GCE)上的大约 100 个容器实例上进行的大规模实验，验证了 Sinan 的可伸缩性，并证明了部署在本地集群上的模型可以在 GCE 上重用，只需进行微小的调整，而不是再训练。

最后，我们展示了 Sinan 模型的可解释性好处，并深入研究了它们可以为大规模系统的设计提供的见解。具体地说，我们使用了 Redis 的日志同步的一个例子，Sinan 帮助将其确定为数十个依赖的微服务中不可预测的性能的来源，以表明该系统可以为那些规模使以前的经验方法不切实际的集群提供实用和深刻的解决方案。

2. 原理和优势

经验资源管理，如自动伸缩或基于排队分析的方法，如 PowerChief，容易出现不可预测的性能或资源低低下。为了解决这些挑战，我们采用了一种数据驱动的方法，从用户那里抽象出微服务的复杂性，并利用 ML 来识别依赖关系对端到端性能的影响，并做出分配决策。我们还设计了一种有效的空间探索算法，针对不同的应用场景探索资源分配空间，特别是可能引入 QoS 违反的边界区域。具体来说，Sinan 的 ML 模型预测了系统的状态和历史，资源配置的端到端延迟和违反 QoS 冲突的概率。该系统利用这些预测来最大化资源效率，同时满足 QoS。

在高层层面上，Sinan 的工作流程如下：数据收集代理收集训练数据，使用一个精心设计的算法来解决挑战（有效地探索资源空间）。根据收集到的数据，Sinan 训练了两个 ML 模型：一个卷积神经网络(CNN)模型和一个增强树(BT)模型。CNN 通过预测在不久的将来的端到端尾延迟来处理层之间的依赖关系和导航系统复杂性。BT 模型通过评估未来 QoS 违反的概率来解决延迟排队效应，以考虑系统在建立队列时的惯性。在运行时，Sinan 推断出瞬时尾部延迟和即将发生的 QoS 违反的概率，并相应地调整资源以满足 QoS 约束。如果应用程序或底层系统在任何时间点发生变化，Sinan 将重新训练相应的模型，以解释这些变化对端到端性能的影响。

Sinan 的 ML 模型的目标是准确地预测给定一定资源分配的应用程序的性能。然后，调度器可以用每个微服务可能的资源分配查询模型，并选择以最

少必要资源满足 QoS 的模型。

实现这一点的一个简单方法是设计一个 ML 模型，它预测直接的端到端尾延迟作为资源分配和利用率的函数，因为 QoS 是根据延迟定义的，并且在部署期间将预测的延迟与测量的延迟进行比较是很简单的。这种方法的警告是在 Sec 中描述的延迟排队效应。因此，一个分配决策的影响只会出现在以后的性能中。作为一种解决方案，我们实验训练一个神经网络(NN)来预测未来时间窗口内的延迟分布：例如，在接下来的 5 秒内每秒的延迟。然而，我们发现预测精度迅速下降进一步到未来 NN 试图预测，预测只是基于收集当前和过去的指标（资源利用率和延迟），这是足够准确的近期预测，但不足以捕捉微服务之间的依赖如何导致性能发展。

考虑到进一步预测未来延迟的困难，我们设置了另一个目标：预测近期的延迟，这样就可以快速识别即将发生的 QoS 违规，但只预测以后经历 QoS 违规的概率，而不是每个决策间隔的确切延迟。与详细的延迟预测相比，这种二进制分类是一个更复杂的问题，并且仍然向资源经理传递了关于性能事件的足够信息，例如，QoS 违反，目前可能需要立即采取行动。

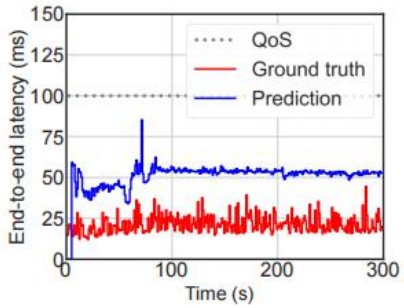


图 1 多任务神经网络过度预测社交网络延迟

一种直观的方法是多任务学习神经网络，它预测下一个间隔的延迟，以及接下来几个间隔内的 QoS 违反概率。然而，多任务神经网络显著高估了尾延迟和 QoS 违反概率，如图 4 所示。请注意，预测和地面真实之间的差距并不表明恒定的差异，这一点具有很强过拟合能力的 NNs 可以很容易地学习到。我们将高估归因于 QoS 违反概率、0 到 1 之间的值和延迟之间的语义差距造成的干扰，这个值不是严格限制的。

为了解决这个问题，我们设计了一个两阶段模型：首先，一个 CNN，它可以高精度地预测下一个时间步长的端到端延迟，其次，一个增强树(BT)模型，使用 CNN 提取的潜在变量，估计未来 QoS 违反的概率。BT 通常比 CNNs 更不容易发生过拟合，因为它的可调超参数比 NNs 要少得多；主要是树的数量和树的深度。通过使用两个独立的模型，Sinan 能够为各自的目标优化每个模型，并避免了对两个任务使用一个联合的、昂贵的模型的过度预测问题。我们将 CNN 模型称为短期延迟预测器，将 BT 模型称为长期违反预测器。

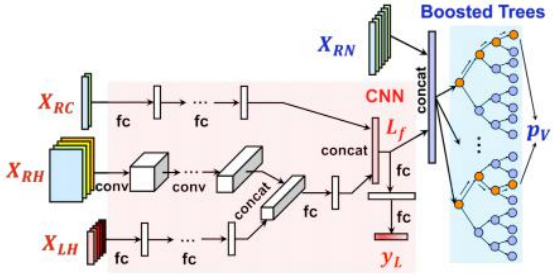


图 2 Sinan 的混合模型，由一个 CNN 和一个增强树(BT)模型组成。

4 研究进展

4.1 系统架构

Sinan 由三个组件组成：一个集中式调度器、部署在每个服务器/VM 上的分布式操作符和一个托管 ML 模型的预测服务。

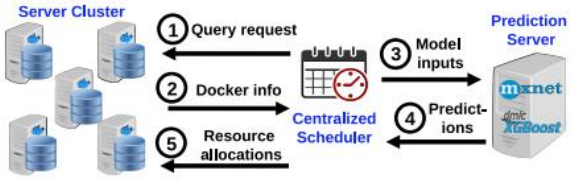


图 3 思南的系统架构。当收到用户请求时，Sinan 通过 Docker 和 Jaeger 收集资源和性能指标，将收集到的指标输入到 ML 模型中，并使用模型的输出相应地为每个层分配资源。系统会定期在线地重新评估分配决策。

思南会定期做出决定。在每个 1s 的决策间隔内(与定义 QoS 的粒度一致)，集中式调度器查询分布式操作符，以获得前一个间隔内每个层的 CPU、内存和网络利用率。资源使用可以从 Docker 的监视基础设施中获得，并且只涉及少量的文件读取，所产生的开销可以忽略不计。除了每层信息外，调度程序还查询 API 网关，以从工作负载生成器获取用户负载统计信息。调度器将此数据发送到混合 ML 模型，该模型负责评估不同资源分配的影响。在作为模型的输入之前，对同一层副本的资源使用量进

行平均。基于模型的输出，Sinan 选择一个使用最少必要资源满足 QoS 的分配向量，并将其决策传递给每个节点的代理进行执行。

Sinan 主要关注计算资源，它对微服务性能的影响最大。Sinan 除了为每个微服务分配多个核外，还探索了子核分配，以避免非资源需求层的资源低低下，并实现更密集的托管。

4.2 资源分配空间探索

具有代表性的训练数据是提高任何 ML 模型的准确性的关键。理想情况下，在在线部署过程中遇到的测试数据应该遵循与训练数据集相同的分布，从而避免协变量的偏移。特别是对于我们的问题，训练数据集需要覆盖在在线部署过程中可能发生的足够范围的应用程序行为。因为 Sinan 试图在不牺牲资源效率的情况下满足 QoS，所以它必须有效地探索资源分配空间的边界，即使用 QoS 下最小资源量的点。我们将数据收集算法设计为一个多武装的强盗过程，其中每一层都是一个独立的臂，目的是最大化资源和端到端 QoS 之间关系的知识。

数据收集算法用一个元组近似应用程序的运行状态，其中是 rps 每秒的输入请求，latcur 是当前的尾延迟，latdiff 是与前一个间隔的尾延迟差异，以捕获消费或累积队列的速率。通过调整其分配的资源，每个层都被视为一个可以独立发挥的手臂。对于每一层，我们近似的映射资源和端到端 QoS 伯努利分布，概率 满足端到端 QoS，我们定义我们的信息从分配一定数量的资源一层，减少预期的置信区间相应的伯努利分布。

$$op_T^s = \arg\max_{op} C_{op} \cdot \left(\sqrt{\frac{p(1-p)}{n}} - p\sqrt{\frac{p_+(1-p_+)}{n+1}} - (1-p)\sqrt{\frac{p_-(1-p_-)}{n+1}}\right)$$

通过选择最大化公式的操作。数据收集算法被鼓励探索以最小资源量满足 QoS 的边界点，因为探索明确满足或违反 QoS 的分配（使用 =1 或 =0）最多可以获得 0 个信息。相反，该算法优先考虑探索那些对 QoS 的影响是不确定性的资源分配，比如那些使用 =0.5 的资源分配。同样值得注意的是，状态编码和信息增益定义是对实际系统的简化近似，其唯一目的是包含感兴趣区域内的勘探过程。最终，我们依赖于 ML 来提取在微服务图中包含层间依赖关系的状态表示。

为了修剪操作空间，Sinan 对数据收集和在线调度都强制执行了一些规则。首先，调度程序只允许从预定义的操作集中进行选择。具体来说，在我们的设置中，操作包括将或增加 CPU 分配减少 0.2 到 1.0CPU，以及增加或将服务的总 CPU 分配减少 10%或 30%。这些比率是根据 AWS 步骤缩放教程[4]来选择的；只要 CPU 分配的粒度不改变，其他资源比率也可以在不重新训练模型的情况下工作。其次，对每一层都强制执行 CPU 利用率的上限，以避免过度激进的资源缩减，从而导致长队列和删除请求。第三，当端到端尾端延迟超过预期值时，Sinan 会禁用资源回收，以便系统能够尽快恢复。与在线部署的一个细微区别是，数据收集算法探索了尾部延迟区域的资源分配，其中与 QoS 相比是一个小值。额外的允许数据收集过程探索导致轻微 QoS 违规的分配，而无需立即恢复到满足 QoS 的状态的压力，这样 ML 模型就可以知道边界情况，并在部署时避免它们。在我们的设置中，经验是 QoS 的 20%，以充分探索分配空间，而不会导致尾部延迟分布偏离部署中可能看到的值。只在系统名义上运行时收集数据，或随机探索分配空间不满足这些要求。

图 4 显示了训练数据集中的延迟分布，以及模型的训练和验证误差如何相对于训练数据集中观察到的延迟范围发生变化。在第二幅图中，x 轴为训练数据集中样本的延迟，左 y 轴为 CNN 的均方根误差 RMSE，右 y 轴表示 XGBoost 的分类错误率。每个点的 y 轴值是模型在只使用延迟小于相应 x 值的数据进行训练时的训练和验证误差。如果训练数据集不包含任何违反 QoS(500ms)的样本，那么 CNN 和 XGBoost 都经历了严重的过拟合，极大地误预测了延迟和 QoS 违反。

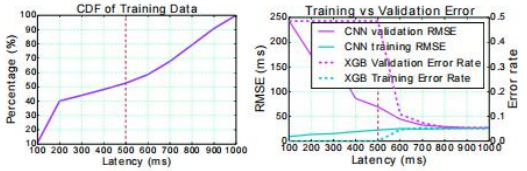
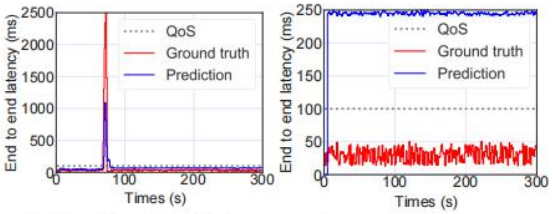


图 4 训练数据集延迟分布和 ML 训练与验证错误相关的数据集延迟范围

图 5 显示了使用不管理数据集分布的数据收集机制收集的数据。当存在自动缩放时（大多数云中的一种常见的资源管理方案）和随机探索资源分配时，收集训练数据集时的预测精度。正如预期的那样，当使用自动缩放时，模型没有看到足够的违反 QoS 的情况，因此严重低估了延迟，并导致尾部延

迟的大峰值，迫使调度器使用所有可用资源来防止进一步的违反。另一方面，当使用随机分析训练模型时，它不断高估延迟并禁止任何资源减少，突出了联合设计数据收集算法和 ML 模型的重要性。



(a) Autoscaling data collection. (b) Random data collection.

图 5 与(a)自动缩放和(b)随机数据收集方案的预测和真延迟的比较。

增量和转移学习：增量再培训可以应用于适应部署策略或微服务更新的变化。在部署过程中，再训练可以在后台或当预测精度低于预期阈值时定期触发。在微服务图的拓扑结构不受影响的情况下，例如硬件更新。随着公共云提供商的变化，微调等迁移学习技术可以用于用新收集的数据在后台训练 ML 模型。如果拓扑被改变，CNN 需要修改以考虑删除和新添加的层。

附加资源：Sinan 可以扩展到其他系统资源。一些资源，如网络带宽和内存容量充当阈值，低于阈值性能显著下降，如网络带宽，或应用程序经历内存错误，并且可以用更简单的模型进行管理，如设置内存使用的固定阈值，或根据网络带宽的用户负载按比例扩展。

4.3 在线调度程序

在部署期间，调度程序使用 ML 模型评估资源分配，并在不进行过度配置的情况下选择满足端到端 QoS 的适当配置。

在线评估所有潜在的资源分配将是非常昂贵的，特别是对于复杂的微服务拓扑。相反，调度程序会按照表 1 中所示的启发式方法集来计算分配的一个子集。对于缩小操作，调度程序评估单层和层批次的 CPU 分配，例如，以最低的 cpu 利用率缩小 k 层。N 表示微服务图中的层数。当需要扩展时，调度程序会检查扩展单个层、所有层或在过去 决策间隔中缩小的层集的影响，1<t<T 采用经验选择 T。最后，调度程序还将评估维护当前资源分配的影响。

调度程序首先排除其预测的尾部延迟高于 QoS-RMSE 的操作。然后利用预测的违反概率过滤掉风险操作，使用两个用户定义的阈值。这些阈值类似于自动缩放中使用的阈值，其中较低的阈值触发缩小，较高的阈值放大；两个阈值之间的区域表示稳定操作，其中保留当前资源分配。具体来说，当保持当前分配的违反概率小于 pu 时，则认为该操作是可接受的。同样，如果存在违反概率低于 pu 的缩小操作，也被认为是可以接受的。当保持操作的违反概率大于 pu 时，只接受违反概率小于 pu 的放大操作；如果没有这样的操作存在，所有层都会扩大到其最大数量。我们设置 pu 验证研究的假否定不超过 1%消除 QoS 违规，和 pd 值小于 pu 有利于稳定的资源分配，所以资源不会波动太频繁，除非有显著波动利用率和/或用户需求。在所有可接受的操作中，调度程序选择需要资源最少的操作。

对于 ML 模型的预测延迟或 QoS 违反概率显著偏离地面真相的情况，调度器也有一个安全机制。如果发生错误预测的 QoS 违反，Sinan 会立即升级所有层的资源。此外，给定模型的信任阈值，当延迟预测错误或错过的 QoS 违反的数量超过阈值时，调度器就会减少其对模型的信任，并在回收资源时变得更加保守。在实践中，Sinan 从不需要降低其对 ML 模型的信任度。

4.4 Sinan 的可扩展性

我们现在展示了思南在运行社交网络的 GCE 网络上的可扩展性。除 CNN 外，XGBoost 的训练和验证准确率分别为 96.1%和 95.0%。该模型的大小和速度保持不变，因为它们与本地集群模型共享相同的体系结构。

为了进一步测试 Sinan 对工作负载变化的稳健性，我们通过不同的请求类型，为社交网络实验了四个工作负载。有些请求，如合成帖子涉及大部分微服务，因此资源密集型，而其他请求，如读者时间线涉及的层要少得多，并且更容易分配资源。我们改变合成帖子：阅读时间线：阅读时间轴请求的比例；w0w1 和 w2 工作负载的比例为 5：80：15、10：80：10、1：90：9 和 5：70：25，其中 w0 的比例与训练集的比例相同。这些比率代表了不同的社交媒体参与场景。Sinan 总是满足 QoS，并相应地调整资源。w1 需要最大的计算资源(450 个用户需要 170 个 vcpu)，因为因为Ω请求的数量最高，这会触发计算密集型的 ML 微服务。

4.4 可解释的机器学习模型

对于用户信任 ML 来说，重要的是要根据它所管理的系统来解释其输出，而不是把 ML 当作一个

黑盒子。我们特别感兴趣的是理解什么使模型中的一些特性比其他特性更重要。优点有三重：1）调试模型；2）识别和修复性能问题；3）过滤虚假特征，减少模型规模，加快推理速度。

可解释性方法。对于 CNN 模型，我们采用了广泛使用的 ML 可解释性方法 LIME。LIME 通过识别对预测贡献最大的关键输入特征来解释神经网络。给定一个输入 X，LIME 扰动 X，得到一组在特征空间中接近 X 的人工样本。然后，LIME 用神经网络对扰动样本进行分类，并利用标记的数据拟合一个线性回归模型。由于线性回归很容易解释，LIME 使用它来识别基于回归参数的重要特征。由于我们主要感兴趣的是理解 QoS 违规的罪魁祸首，我们从发生 QoS 违规发生的时间步中选择样本 X。我们通过将给定层或资源的特征乘以不同的常数来干扰给定层或资源的特征。例如，为了研究 MongoDB 的重要性，我们将其利用历史用两个常数 0.5 和 0.7 相乘，并生成多个扰动样本。然后，我们构建一个包含所有扰动和原始数据的数据集来训练线性回归模型。最后，我们通过对每个特征的相关权重的重要性进行排序。

解释 CNN:我们使用 LIME 来纠正社交网络中的性能问题，尽管负载较低，但尾部延迟仍经历了峰值和不稳定期.手动调试很麻烦，因为它需要深入研究每个层，以及可能的层组合来确定根本原因。相反，我们利用可解释的 ML 来过滤搜索空间。首先，我们确定了最重要的 5 层,我们发现，模型预测最重要的层是社交图 Redis，而不是像 nginx 这样 CPU 利用率很高的层。

然后，我们检查了每个资源度量对 Redis 的重要性，发现最有意义的资源是缓存和驻留工作集大小，它们对应于内存中的磁盘的数据，以及非缓存的内存，包括堆栈和堆。使用这些提示，我们检查 Redis 的内存配置和统计信息，并确定它被设置为每分钟在持久存储中记录日志。对于每个操作，Redis 分叉一个新进程并将所有写入的内存复制到磁盘；在此期间，它停止服务请求。

禁用日志持久性消除了大部分延迟峰值我们进一步分析功能模型的重要性训练数据修改后的社交网络，并发现社交图 Redis 的重要性显著降低。

4.5 相关工作

微服务：微服务的出现促使了最近研究其特点和系统含义的工作。死亡之星平台包括几个使用微服务构建的端到端应用程序，并从服务器设计、网络和操作系统开销、集群管理、编程框架和规模影响的细节等方面探讨了微服务的系统影响。uSuite 还引入了许多使用微服务构建的多层应用程序，并研究了它们的性能和资源特性。乌尔冈卡尔等人。将分析建模引入多层应用程序，准确地捕捉了并发限制和缓存策略等方面的影响。所有这些研究的结论是，尽管微服务有好处，但它们改变了当前云基础设施设计的几个假设，在硬件和软件方面引入了新的系统挑战。

在资源管理方面，微信通过匹配上下游服务的吞吐量，通过过载控制管理微服务；动态提升多阶段应用中的瓶颈服务，Suresh 等。利用过载控制，并采用基于最后期限的调度来提高多层工作负载中的尾部延迟。最后，斯里拉曼等人。为微服务并发性提供了一个自动调优框架，并显示了线程决策对应用程序性能和响应性的影响。

云资源管理：云计算的流行推动了许多集群管理设计。类星体、Mesos、Torque 和 Omega 都是大型、多租户集群中的目标资源分配。类星体是一个感知 qos 的集群管理器，它利用机器学习来识别新的、未知的应用程序的资源偏好，并在不牺牲资源效率的情况下以满足其性能需求的方式分配资源。Mesos 是一个两级调度程序，它为不同的租户提供资源，而 Omega 使用共享状态方法来扩展到更大的集群。最近，各方利用了这样一种直觉，即资源可以通过资源分区在服务器上共同定位多个交互式服务。自动缩放是基于利用率的弹性缩放分配的行业标准。虽然所有这些系统都提高了它们所管理的云基础设施的性能和/或资源效率，但它们是为单片应用程序或具有几层的服务设计的，不能直接应用于微服务。

云系统中的 ML：人们对利用 ML 来解决系统问题，特别是资源管理越来越感兴趣。类星体利用协同过滤来识别未知作业的适当资源分配。自动驾驶仪使用一系列模型来推断高效的 CPU 和内存作业配置。资源中心表征 VM 实例的行为，并离线训练一组 ML 模型，它们可以准确地预测 CPU 利用率、部署大小、生命周期等。使用随机的森林和促进树木的生长。最后，Seer 提出了一个针对微服务的性能调试系统，该系统利用深度学习来识别常见性能问题的模式，并帮助定位和解决它们。

5 总结与展望

我们介绍了 Sinan，一个可伸缩的和 qos 感知的交互式微服务资源管理器。Sinan 强调了管理复杂微服务的挑战，并利用一组经过验证的 ML 模型来推断分配对端到端尾部延迟的影响。Sinan 负责在线运营，并调整其决策，以应对应用程序的变化。我们对不同微服务下的本地云和公共云)进行了评估，结果表明它在不牺牲资源效率的情况下满足 QoS。Sinan 强调了自动化的、数据驱动的方法的重要性，即以一种实际的方式管理云的复杂性。

延迟预测器将资源使用历史(XRH)、延迟历史(XLH)和潜在资源配置(XRC)作为输入，并预测下一个时间步的端到端尾延迟分布(yL)

违规预测器处理一个二元分类任务，即给定的分配是否会在将来导致 QoS 违规，以过滤掉不需要的资源选项。我们使用 XGBoost，它通过组合一系列简单的回归树来实现精确的非线性模型。它将目标建模为树的和，每个树将特征映射到一个分数。最终的预测是通过在所有树中积累分数来确定的。

、