第 13 卷 第 6 期
2024 年 11 月
集 成 技 术
JOURNAL OF INTEGRATION TECHNOLOGY
Vol. 13 No. 6
Nov. 2024

**引文格式：**
陈文雄, 李乐乐, 喻之斌. 基于 Linux 系统参数在 X86 与 ARM 架构下 Nginx 性能调优的比较分析 [J]. 集成技术, 2024, 13(6): 16-30.
Chen WX, Li LL, Yu ZB. Comparative analysis of Nginx performance tuning based on Linux system parameters on X86 versus ARM architectures [J]. Journal of Integration Technology, 2024, 13(6): 16-30.

# 基于 Linux 系统参数在 X86 与 ARM 架构下 Nginx 性能调优的比较分析

陈文雄[1,2] 李乐乐[1] 喻之斌[1,2*]

[1]（中国科学院深圳先进技术研究院 深圳 518055）
[2]（中国科学院大学 北京 100049）

**摘 要** 在当今的数字时代，Nginx 已成为 Linux 系统上最为普及的 Web 应用服务器，占据的市场份额排第一名。由于 Nginx 在确保用户服务质量方面的作用较关键，因此，对 Nginx 性能的优化显得尤为重要。Nginx 服务器虽然广泛部署于 X86 和 ARM 两种主要的硬件架构上，但迄今为止，针对这两种架构下 Nginx 性能调优的对比分析尚为空白。该研究旨在填补这一缺口，通过对比这两种架构的系统参数自动调优效果，明确指出 X86 和 ARM 在不同场景（动态与静态请求处理）的性能差异：在动态请求处理场景中，X86 架构的第 99 百分位延迟比 ARM 低 515 ms，性能提升高达 287%。而在静态请求处理时，ARM 架构的第 99 百分位延迟比 X86 低 220 ms，性能提升高达 60%。这一发现突出了 X86 和 ARM 架构在处理不同负载时的特定优势，并明确指出了不同硬件架构对 Nginx 性能优化策略的显著影响。因此，系统管理员在针对不同硬件架构进行 Nginx 性能优化时，必须考量架构特有的静态与动态请求之间的性能差异和迭代效率，以确保最佳性能表现。

**关键词** Nginx；X86；ARM；Linux 参数；Nginx 性能调优
**中图分类号** TP302.7 **文献标志码** A doi: 10.12146/j.issn.2095-3135.20240307002

**收稿日期**：2024-03-07 **修回日期**：2024-04-11
**基金项目**：深圳市科技计划项目（JCYJ20220818101607015）
**作者简介**：陈文雄，硕士研究生，研究方向为 Nginx Web 服务器的性能优化等；李乐乐，科研助理，研究方向为云计算资源优化、大数据性能加速等；喻之斌（通讯作者），博士，研究员，博士研究生导师，研究方向为多核体系结构、通用 GPU 体系结构、大数据体系结构、云计算等，E-mail：zb.yu@siat.ac.cn。

# Comparative Analysis of Nginx Performance Tuning Based on Linux System Parameters on X86 versus ARM Architectures

CHEN Wenxiong[1,2]　LI Lele[1]　YU Zhibin[1,2*]

[1]( *Shenzhen Institute of Advanced Technology*, *Chinese Academy of Sciences*, *Shenzhen* 518055, *China* )

[2]( *University of Chinese Academy of Sciences*, *Beijing* 100049, *China* )

[*]*Corresponding Author*: *zb.yu@siat.ac.cn*

**Abstract**　In today's digital age, Nginx has become the most widely used Web application server on Linux systems, holding the top position in market share. Nginx plays a critical role in ensuring user service quality, making its performance optimization crucial. Although Nginx servers are widely deployed on both X86 and ARM architectures, there is a lack of comparative analysis on performance tuning for these architectures. This study aims to fill this gap by comparing automatic system parameter tuning on Nginx across the two architectures. It identifies the performance differences between X86 and ARM in different scenarios (dynamic and static request processing). When handling dynamic requests, Nginx on the X86 architecture achieves a 99th percentile latency 515 ms lower than that on the ARM architecture, reflecting a performance improvement of 287%. Conversely, in static request processing, the ARM architecture performs better, with a 99th percentile latency 220 ms lower than that of X86, marking a 60% performance increase. These findings highlight the distinct advantages of X86 and ARM architectures in handling different loads and the significant impact of hardware architecture on Nginx performance optimization strategies. Therefore, system administrators must consider performance differences between static and dynamic requests and the unique characteristics of each architecture to achieve optimal performance.

**Keywords**　Nginx; X86; ARM; Linux parameters; Nginx performance tuning

**Funding**　This work is supported by Shenzhen Science and Technology Program (JCYJ20220818101607015)

## 1　引　言

随着互联网技术的不断进步和应用的日益复杂化，对 Web 服务器的性能要求也相应提升。Nginx[1]是一款在全球范围内被广泛采用的高性能 Web 服务器，因优异的并发处理能力和资源利用效率而受到业界的广泛认可，根据 W3Techs 排名[2]，Nginx 在世界前 100 000 个网站中占据了 34.1% 的市场份额，稳居第一。这种服务器软件能有效支持高流量的 Web 服务，可在用户访问高峰期间，确保网站的快速响应和稳定运行[3]。Nginx 的性能不仅取决于其自身的优化和配置，还受运行它的硬件平台特性的影响[4-5]。当前，X86 和 ARM 架构是市场上最主流的两种服务器硬件架构，它们在设计理念、能效比和适用场景上存在显著差异。X86 架构以其强大的计算能力和广泛的应用支持在服务器市场上占据主导地位[6]，而 ARM 架构则以其高效的能源利用率和较低的成本在某些领域逐渐获得优势[7]。随着 ARM 架构服务器性能的不断提升，它们开始被越来越多地用于运行网络服务和应用，与传统的 X86 服务器形成了竞争。

Nginx 虽然已在 X86 和 ARM 两种硬件架构上广泛部署，但对这些架构上的性能表现进行系统性比较的研究还相对缺乏。特别地，随着 ARM 架构在高性能服务器市场上的崛起，这一研究空白显得更为重要。近年来，ARM 架构已开始应用于 Nginx 服务器中，并展现出了不错的效果。特别是在亚马逊的 EC2 AMI 中，基于 ARM 架构的 Graviton2 处理器支持的 Nginx 运行效率更高，相比于相同环境中的 X86 实例，ARM 实例的性价比提高了 40%，成本降低了 20%[8]。这一发现不仅证实了 ARM 架构服务器在运行 Nginx 方面的潜力，也突显了深入了解 Nginx 在 X86 与 ARM 架构上性能差异的重要性，这对系统管理员选择最合适的硬件平台和优化服务器配置具有重要的实际意义。

此外，Linux 系统参数的调整是提升服务器性能的关键，直接影响操作系统级别的资源分配和任务调度，如内存管理、网络连接和文件系统设置，间接影响运行在其上的应用程序性能，包括 Nginx 的运行表现。常见的 Linux 系统参数如 net.core.somaxconn（控制连接队列长度）和 vm.swappiness（管理内存交换）等，显著影响服务器在高并发或资源紧张条件下的表现[9]。然而，关于这些参数调整在不同硬件架构下对 Nginx 性能的具体影响，现有研究成果相对有限，尤其缺乏系统化的分析和实验验证。因此，本研究致力于通过系统的实验评估和分析，填补这一研究空白。

本研究深入探讨了 Linux 系统参数调优对 X86 和 ARM 架构服务器上，Nginx 处理静态和动态请求性能的影响，旨在全面评估和比较两种架构下 Nginx 的性能表现，并基于实验结果识别最有效的系统参数调整策略。本文期望通过上述研究为系统管理员提供科学的配置建议和实践指导，帮助他们在多样化的硬件环境中优化 Nginx 的性能，从而更好地满足用户需求，推动 Web 服务技术的发展。

现有文献回顾表明，尽管 Web 服务器性能优化的研究逐渐增多，但大部分研究侧重于软件层面的优化，如负载均衡算法改进[10]和缓存机制改进[11]等，对硬件架构差异和操作系统参数调优的探讨较少。特别地，ARM 架构作为高性能计算的新选择，在性能优化方面的研究不仅具有重要的现实意义，也有着广泛的应用价值。本研究深入分析了 Linux 系统参数在不同架构下对 Nginx 性能的影响，旨在为该领域提供有价值的见解和理论依据。

本研究系统地比较了 X86 与 ARM 架构服务器上 Nginx 的性能差异，通过一系列细致的实验设计和方法论，揭示不同 Linux 系统参数调整对两种架构性能影响的具体机制，可为系统管理员在面对多变的硬件选项和配置挑战时，提供基于实验数据的直接指导。此外，本研究结果还可为未来在不同硬件平台上部署和优化 Web 服务器的研究提供参考与指导。

## 2 研究现状

### 2.1 X86 架构与 ARM 架构的性能对比

在计算技术领域的现代发展轨迹中，X86 与 ARM 架构的性能比较已成为研究的焦点，这两种架构体现了计算硬件不同的设计理念和应用领域。X86 架构采用复杂指令集计算机（complex instruction set computer，CISC）架构，旨在通过复杂指令集提升任务处理的灵活性和处理能力，广泛应用于服务器和桌面计算环境，尤其适合高性能计算[6]。相反，ARM 架构基于精简指令集计算机（reduced instruction set computer，RISC）架构，其优势在于低功耗和简化的指令集，最早用于移动设备和嵌入式系统[7]。随着技术的演进和市场需求的变化，ARM 架构已逐步扩展到服务器市场，引发了对其性能、能效及应用适宜性的广泛讨论。

目前，对这两种架构的比较虽然已经引起了广泛的关注，但在性能优化方面，特别是在高性能 Web 服务器应用程序，如 Nginx 的运行环境下，关于它们性能差异的研究仍相对有限。近期的研究结果揭示了特定场景下，两种架构的性能和能耗差异。例如，da Silva 等[12]对比了 ARM Cortex AS7 与英特尔 X86 平台(代表为 Nvidia Tegra X1 和 Pentium Braswell N3700)在运行 Apache 和 Nginx 服务器时的性能，发现 ARM 处理器在某些条件下性能略优，但英特尔 CPU 在能耗方面表现更佳。此外，Kalyanasundaram 等[13]的研究结果表明，在大数据处理场景下，ARM 架构服务器能够在保持与 X86 架构服务器相近性能的同时，显著降低运营成本。

此外，Chen 等[14]在 AWS Lambda 上使用 ARM64 处理器托管无服务器功能进行实验，结果表明，尽管只有部分函数在 ARM64 上运行速度更快，但绝大多数函数的托管成本更低。Jiang 等[15]评估了 Amazon Graviton ARM 架构处理器在处理大规模计算密集型工作负载时的能力，并将其与 X86 架构处理器进行了比较。尽管存在一些限制，如 L3 缓存和内存访问速度等，但 ARM 架构处理器在处理 Web 服务、视频转码和 TB 级排序等任务时，能够实现与 X86 架构处理器相当，甚至更高的性价比。

这些研究为理解 X86 与 ARM 架构的性能差异提供了参考，但它们通常集中在特定的测试条件或应用场景上，缺乏在 X86 和 ARM 架构下对 Web 服务器性能优化的全面比较。此外，这些研究通常未深入探讨不同架构的技术细节，例如缓存架构和内存管理策略等因素如何影响 Web 服务的性能和响应能力。同时，不同架构下针对 Nginx 参数优化的研究也相对缺乏。

### 2.2　Nginx 性能调优的研究现状

Nginx 性能调优的研究表明，系统资源(如 CPU、内存和网络带宽等)在处理高并发请求时，对 Nginx 服务器性能的影响显著[16-17]。为提高 Nginx 服务的响应能力和处理效率，有效地管理和优化系统资源是关键。

Wang 等[16]通过对 Nginx 架构和工作流程的深度分析，识别了 Nginx 在初次安装时的性能短板，并通过手动调整 Linux 内核参数，成功提升了 Web 服务器的性能。然而，这种方法在有限的参数调整范围和手动配置过程的复杂性方面存在挑战。Wang 等[18]采用了一种增强的自动调优工具，并在商业和工业服务应用中实施，使 Nginx 性能提升了 2%～14%。这种方法虽然有效，但其参数选择依赖于领域知识和经验，缺乏一种系统化的框架来确定对 Nginx 性能有显著影响的参数组合。Chen 等[19]提出的 INTOP 自动调优策略代表了该领域的一个重要进步，通过系统化识别和调整影响 Nginx 性能的关键参数组合，提供了一种高效且系统化的性能优化策略。这种方法减少了对专业知识的依赖，并通过自动化过程大幅提高了优化的效率和效果。

综上所述，本研究旨在通过深入分析和系统化实验评估填补上述研究空白。此外，本研究专注于评估 Linux 系统参数调整在 X86 与 ARM 架构服务器上运行 Nginx 时的性能影响，旨在全面比较这两种架构在性能优化方面的差异。同时，本研究旨在为系统管理员提供明确的指导，帮助他们在多样化的硬件环境中优化 Nginx 的性能，从而更好地满足用户需求，并为计算行业的可持续发展提供支持。

## 3　方　法

为构建一个详尽且逻辑清晰的实验方法，本研究将深入探讨实验环境设置、测试方法和自动调优方法，以确保研究的科学性和实用性。

### 3.1　实验环境设置

本研究搭建了两套实验集群，每套由 5 台

服务器组成。X86 实验集群配备了 Intel(R) Xeon(R) Silver 4214 CPU，每台服务器有 48 个逻辑核和 125 GB DDR4 内存，共 240 个 CPU 逻辑内核和 625 GB 内存。而 ARM 实验集群在 Nginx 负载均衡服务器上采用 HiSilicon Kunpeng 920-4826 CPU，有 48 个物理核和 124 GB DDR4 内存，其余服务器配置与 X86 实验集群相同。所有服务器运行在 Ubuntu 18.04.6 LTS 操作系统上，Nginx 版本统一为 1.20.1。通过这一详细的实验环境设置，本研究计划系统评估和分析调整 Linux 系统参数后，Nginx 在处理请求时，在这两种硬件平台上的性能差异。

### 3.2 测试方法

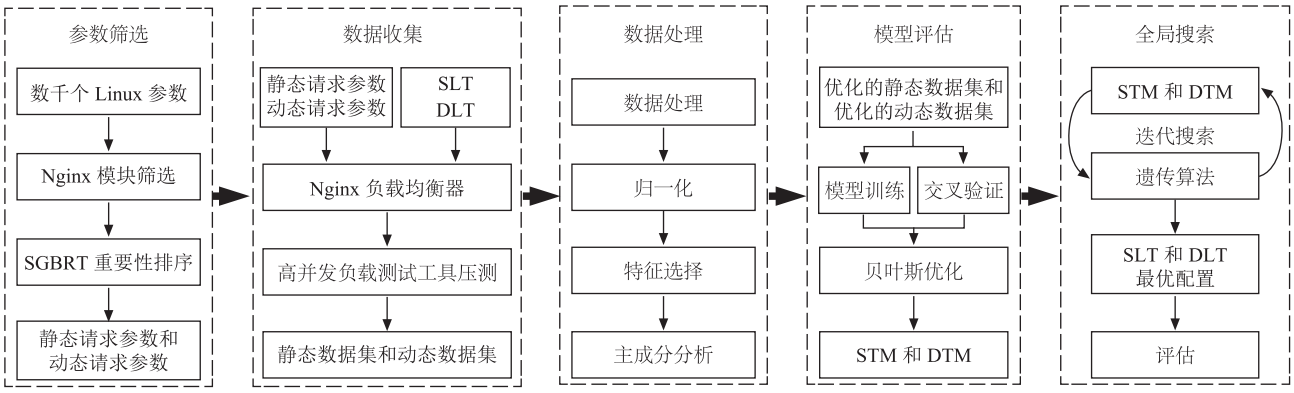
本研究利用 Lua 脚本语言模拟 Nginx 服务器上的静态和动态请求负载[20]，分别称为静态负载测试(static load test，SLT)和动态负载测试(dynamic load test，DLT)。(1)SLT 脚本从预定义的静态文件集中随机选择路径，并将 HTTP GET 请求发到这些路径，模拟了用户在请求网页静态资源时的行为，包括 HTML、CSS、JavaScript 文件、图片和其他资源。(2)DLT 脚本在每次迭代时生成唯一的随机数作为请求参数，进而生成动态统一资源定位符请求，用于模拟用户从网页获取动态内容的过程。这些内容包括通过应用程序编程接口(API)检索的数据和用户提交的表单信息等。SLT 和 DLT 均通过 wrk2(一个开源的 HTTP 压力测试工具)[21]实施。在实验中，配置 wrk2 为以每秒 10 000 个请求的速率发送请求，并通过 Nginx 负载均衡器执行负载测试。这样设置的请求速率旨在模拟高并发场景，测试 Nginx 在不同负载条件下的性能表现。测试的结果，特别是 99% (99th percentile，P99)的延迟，用作评估 Nginx 在 SLT 和 DLT 下性能的实用指标，能直观反映用户的实际体验，虽然 CPU 和内存使用率也是关键的参考指标，但这种测试方法不仅确保了数据的准确性，还有效模拟了真实世界中的 Web 服务请求场景。

### 3.3 自动调优方法

本研究利用 INTOP 方法[19]，通过系统性地调整和评估 Linux 系统参数，优化 Nginx 在 X86 和 ARM 架构服务器上的性能。如图 1 所示，本研究方法涵盖了从参数筛选到全局搜索的 5 个细致步骤，旨在全面提升 Nginx 服务器的性能表现。

#### 3.3.1 参数筛选

首先，本研究细致地研究了 Nginx 处理请求的内部机制，并将这些信息与系统参数进行了关联。其次，通过深入分析 Nginx 的核心模块：




STM：静态数据集训练好的模型(static trained model)；DTM：动态数据集训练好的模型(dynamic trained model)

**图 1 自动调优方法框图**

**Fig. 1 Block diagram of the auto-tuning method**

事件模块、HTTP 模块、邮件模块及其关键的 HTTP 子模块（如 Gzip 和 Cache 子模块）[1]，本研究初步识别出 100 个关键系统参数。特别地，本研究关注了这些模块如何与系统资源（如内存、网络配置）交互[17,22]，并基于其对 Nginx 处理静态和动态请求的性能影响进行筛选。再次，为突出那些对 Nginx 性能有显著影响的参数，本研究利用随机梯度提升回归树[23]进行特征重要性排序。在处理复杂数据集时，随机梯度提升回归树呈现出强大特征选择和评估能力，具有高效性和鲁棒性，因此被广泛应用于对重要参数的此类分析。最后，本研究筛选并确定了最具影响力的静态请求参数和动态请求参数。

3.3.2 数据收集

数据收集阶段旨在系统地记录 Nginx 在不同静态请求参数和动态请求参数配置下的性能指标。通过使用 wrk2 工具执行 SLT 和 DLT，本研究获得了大量性能数据，如 P99 延迟值，为后续的建模阶段提供了丰富的数据集，分别命名为静态数据集和动态数据集。

3.3.3 数据处理

在数据处理阶段，本研究采用了多种数据预处理技术，包括归一化处理、特征选择和主成分分析[24]，旨在提高性能预测模型的准确性和效率。具体来说，归一化处理保证了不同性能指标在模型中的均衡贡献；特征选择帮助去除了无关或冗余的特征，降低了模型复杂度；主成分分析通过降维处理，从海量性能指标中提炼出最关键的因素，为构建高效的预测模型打下了基础。

3.3.4 模型评估

本研究利用收集和处理过的数据，以及深度神经网络（deep neural network，DNN）[25]、支持向量回归（support vector regression，SVR）[26]、随机森林回归（random forest regressor，RFR）[27]、类别提升（category boosting，CatBoost）[28]、轻量级梯度提升机（light gradient boosting machine，LightGBM）[29]和极端梯度提升（extreme gradient boosting，XGBoost）[30]等多种机器学习算法构建性能预测模型，旨在建立系统参数配置与 Nginx 性能（延迟）之间的精确关系。模型的评估通过交叉验证[31]进行，以确保模型的泛化能力。同时，利用贝叶斯优化技术[32]微调模型的超参数，确保高准确性的性能预测，形成 STM 和 DTM。

3.3.5 全局搜索

在最终步骤，本研究利用遗传算法（genetic algorithm，GA）[33]结合先前的 STM 和 DTM 执行全局参数搜索。此过程的核心目标是识别出能显著提升 Nginx 在 SLT 和 DLT 中性能的最佳系统参数配置。在面对复杂的优化问题时，GA 展现出卓越的全局搜索能力和避免陷入局部最优解的能力，因此，本研究将 GA 作为参数搜索的工具。此外，现存的众多算法，如模拟退火算法[34]、粒子群优化算法[35]和差分进化算法[36]等，虽也用于探索复杂配置空间，但它们或显示出缓慢的收敛速度，或存在陷入局部最优解的风险。相比之下，GA 通过模拟自然界的进化和遗传机制，在解决包含多个局部最优解的复杂问题方面表现更出色。尤其是，GA 在多个局部最优解之间的搜索鲁棒性使其能在庞大且充满挑战的配置参数空间中精确定位全局最优解。因此，针对本课题中复杂且充满局部最优解的参数空间，本研究选择使用 GA，以识别出能够最大化 Nginx 性能的最佳配置。

本研究方法的设计和执行全面覆盖了 X86 和 ARM 硬件平台，每个实验步骤都独立地在两种平台上进行，以确保实验数据的全面性和精确性。这种系统化和全面的实验设计不仅深入探讨了 Nginx 性能优化的有效方法，而且成功揭示了 X86 和 ARM 架构在性能优化过程中的不同表现和原因。通过精确的参数筛选、全面的数据收集与处理、先进的模型评估方法和有效的全局优化搜索策略，本研究不仅提高了参数调优的可行性

和可靠性，而且为在不同硬件平台上部署和优化 Web 服务器提供了深刻的见解和有效的策略。

## 4 结果与分析

本研究主要关注模型精确度、遗传算法的迭代收敛和性能提升对比 3 个方面。

### 4.1 模型精确度

本研究对 6 种不同的回归模型进行了全面评估，以确定它们在预测 Nginx 服务器性能方面的效果。

具体的模型误差如图 2 所示，通过深入对比分析，本研究观察到，XGBoost、LightGBM、CatBoost 和 RFR 这 4 种集成学习模型在 X86 架构服务器上的平均预测误差分别为 16%、15%、17% 和 19%，而在 ARM 架构服务器上，相应的平均预测误差则分别为 15%、15%、12% 和 16%。这 4 个模型的预测误差都小于 20%，且在 X86 和 ARM 两种硬件平台上，模型的误差差别较小，显示了这些集成学习模型在不同硬件平台上的稳定性和准确性。

相对而言，DNN 和 SVR 模型在 X86 服务器上的平均误差分别达到了 25% 和 26%，而在 ARM 服务器上，这些误差稍有下降，分别为 23% 和 22%。由此表明，与 DNN 和 SVR 相比，XGBoost、LightGBM、CatBoost 和 RFR 的预测精度较好，跨平台应用的误差低、稳定性高。

基于上述分析，本研究选择误差较低的 XGBoost、LightGBM、CatBoost 和 RFR 模型作为 GA 调用的模型(即 STM 和 DTM)。这一选择不仅基于这些模型在 X86 和 ARM 平台上所展示的准确性，还基于它们在处理不同架构服务器数据时的稳定性和鲁棒性[37]。上述分析结果表明了集成学习模型在预测 Nginx 服务器性能方面的高效性，同时强调在进行硬件平台选择时，模型的准确性和稳定性是两个重要的考量因素。

### 4.2 遗传算法的迭代收敛

本研究利用 GA 深入探索操作系统参数配置，以确定高并发负载测试下，Nginx 的最优配置。同时，本实验旨在探讨如何有效减少寻找最优配置所需的迭代次数和时间。

#### 4.2.1 遗传算法的迭代收敛次数

GA 的迭代收敛次数如图 3 所示，在 SLT 和 DLT 中，与 X86 架构相比，ARM 架构所需的迭代次数显著减少，分别从 34～77 次降至 22～54 次。这主要归因于 ARM 架构的设计原则和能效优化。ARM 的 RISC 架构设计原则[7]使其在执行 GA 搜索时，因更简洁的指令执行流程而更高

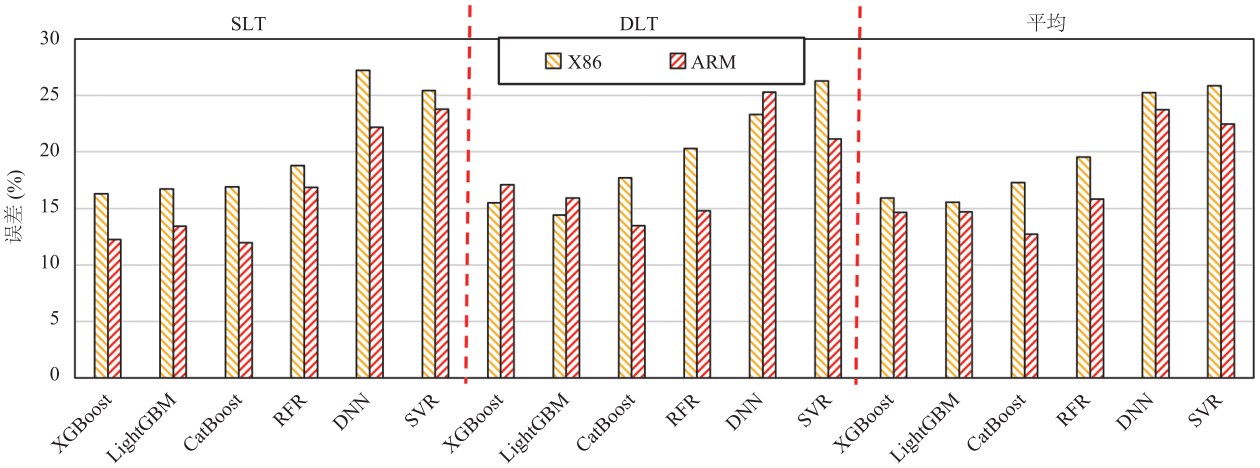



图 2 模型误差

Fig. 2 Model error

效，特别是在处理遗传算法等需要大量简单迭代操作的场景中。此外，ARM 架构的能效优化特性可能在 GA 迭代过程中的资源管理方面产生积极影响[7]，使 ARM 服务器在追求最优配置过程中能维持更高的运算效率。这两点因素共同导致 ARM 架构在寻找最优参数配置时需要更少的迭代次数。

此外，本实验还发现，负载测试的类型对迭代效率具有重要影响。如图 3(a)～(b)所示，在 X86 架构中，采用 XGBoost 模型进行 SLT 训练所需的迭代次数为 35，而进行 DLT 训练则需 60 次；相应地，如图 3(c)～(d)所示，在 ARM 架构中，SLT 和 DLT 训练的迭代次数分别为 23 和 44。由此表明，优化过程中，架构和请求类型对迭代效率有显著影响。此外，SLT 和 DLT 训练所需迭代次数的差异还反映了不同请求类型的影响。由于 SLT 通常涉及对固定资源的请求，其数据复杂性相对较低，因此使得集成学习模型，如 XGBoost、LightGBM、CatBoost 和 RFR 等，能更快地识别出影响性能的关键特征，并快速调整参数，减少所需的迭代次数。相比之下，DLT 的动态内容增加了数据的复杂性，导致模型在适应和学习这种复杂性时需迭代更多次。

4.2.2　遗传算法的迭代收敛时间

GA 的迭代时间如图 4 所示，ARM 架构在迭代次数方面虽然比 X86 架构有优势，但这并不直接导致其迭代时间总体较短。具体地，在 SLT 中，ARM 架构结合 LightGBM 模型寻找最优配置的时间仅为 2 min，而 X86 架构需要 6 min。相反，在 DLT 中，X86 架构使用 LightGBM 模型找到最优配置的时间减少至 4 min，而 ARM 架构所需时间延长至 15 min。这些结果揭示了两种架构在处理 Nginx 的不同负载测试方面的明显差异：ARM 架构在处理静态负载测试时展现出显著的效率优势，而 X86 架构在动态负载测试中表现更为高效。

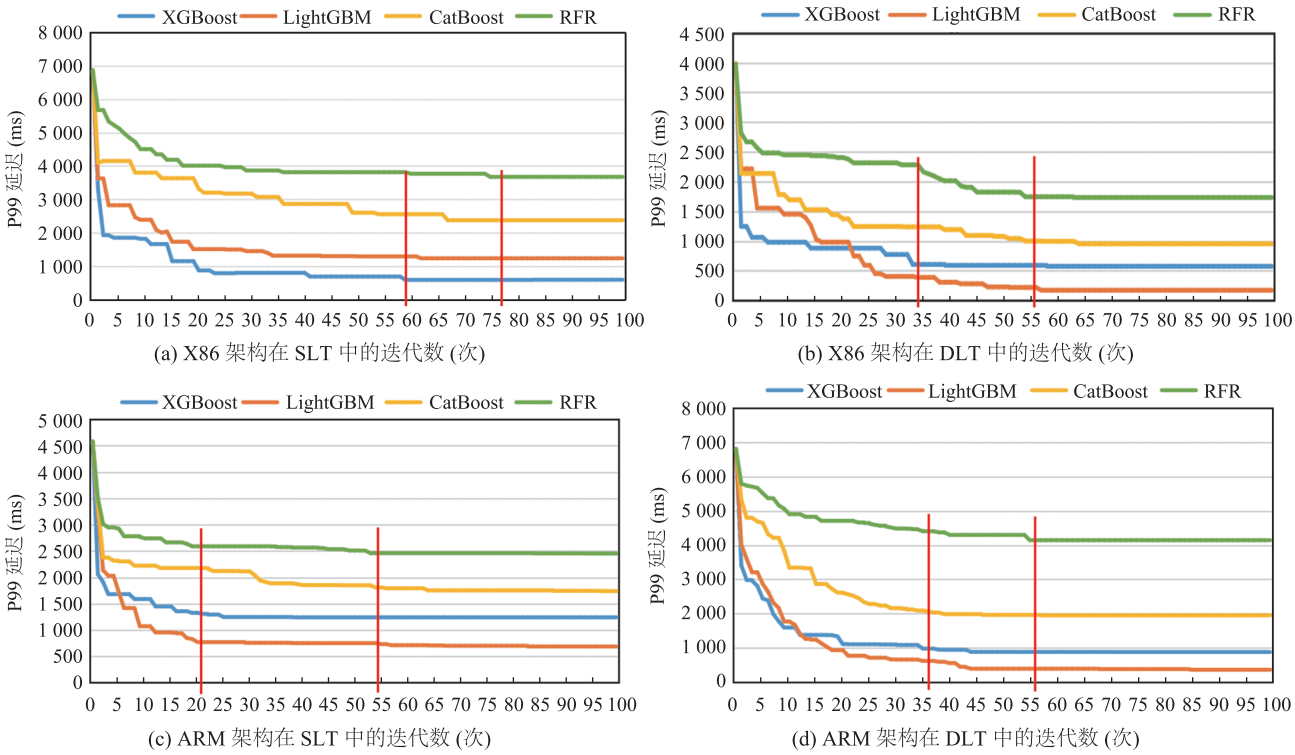



**图 3　X86 和 ARM 架构在静态负载测试(SLT)和动态负载测试(DLT)中的迭代次数**

**Fig. 3　The number of iterations in static load test (SLT) and dynamic load test (DLT) for X86 and ARM architectures**

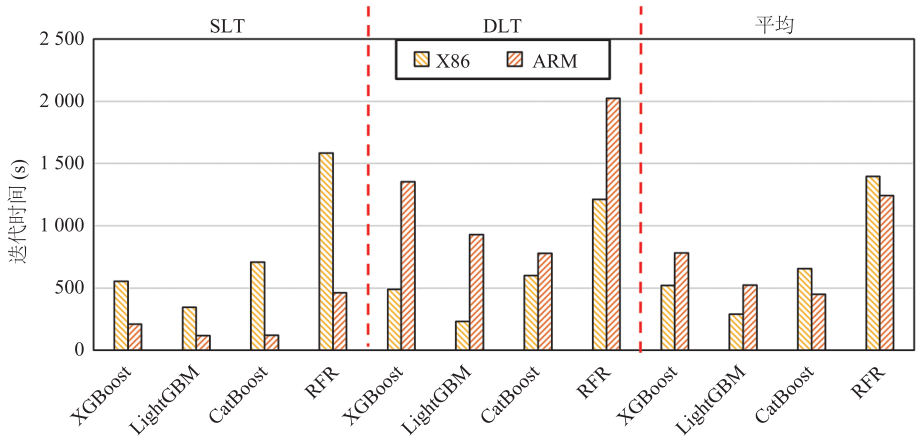



**图 4 遗传算法的迭代时间**

**Fig. 4 Iteration time of the genetic algorithm**

上述差异的原因如下：

(1) 架构设计特点：ARM 架构的 RISC 架构优化了简单重复任务的处理效率[7]，这对 SLT 中相对简单、重复的请求处理有利。相比之下，X86 架构的 CISC 架构更适合处理复杂的计算任务[6]，如 DLT 中的动态内容生成和处理。

(2) 系统资源管理：ARM 架构在能效管理方面的优化可使其在执行简单、计算不密集的任务时更高效[7]，从而在静态负载测试中快速收敛。而 X86 架构具有强大的计算资源和优化的资源调度机制[6]，在处理资源密集型的动态请求时能更快地找到最优配置。

(3) 负载测试特性：SLT 通常涉及对固定资源的请求[20]，其性能优化较为直接和简单；而 DLT 涉及更多的逻辑处理和资源动态分配[20]，其优化过程更为复杂、耗时。

### 4.3 性能提升对比

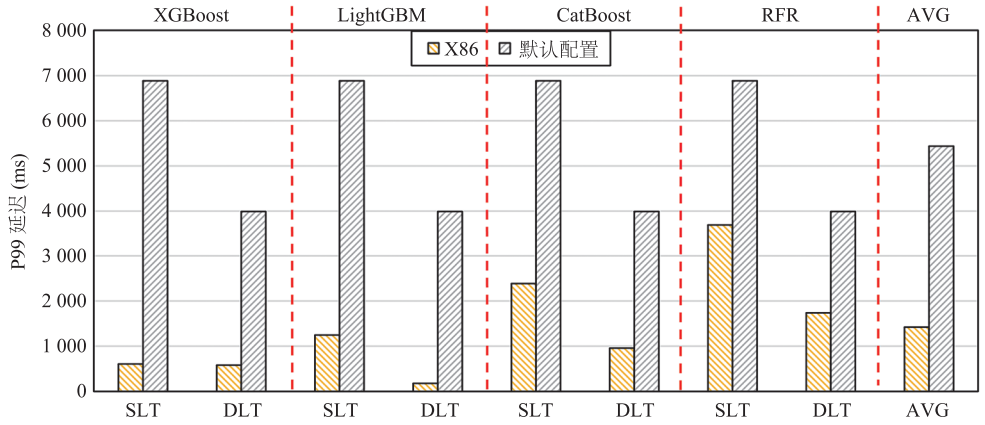
本研究专注于通过 GA 对操作系统参数进行自动调优，目标是在高并发负载测试条件下为 Nginx 实现最优配置。本研究评估了自动调优对 X86 和 ARM 架构下 Nginx 性能的影响。测试场景包括 SLT 和 DLT。

如图 5 所示，自动调优后，X86 架构的性能平均提升了 3.8 倍，最高提升了 22 倍；ARM 架构的性能平均提升了 3.3 倍，最高提升了 12 倍。由此表明，默认配置在资源分配和利用上存在显著的局限性。

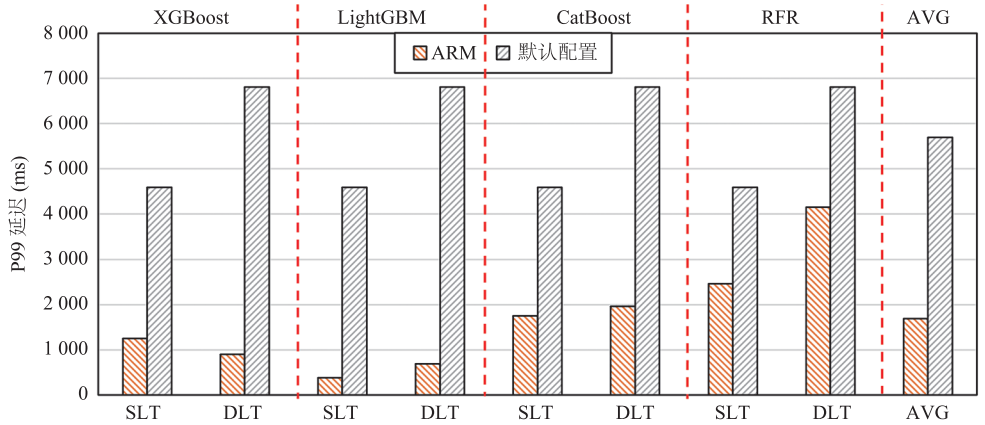
如图 6 所示，在 SLT 中，总体上，ARM 的 P99 延迟比 X86 低，平均减少了 220 ms，Nginx 性能提升幅度达 60%，主要得益于 ARM 架构在处理静态内容时的高能效比和资源利用效率[7]。综上所述，ARM 架构的设计原则优化了能效和处理速度，使其在静态负载测试下表现出较好的性能。

在 DLT 中，X86 架构的 P99 延迟显著低于 ARM 架构，平均减少了 515 ms，Nginx 性能提升幅度达 287%。这种显著差异的原因是，X86 架构在处理动态内容请求时，计算能力更强，并发处理机制更优[6]。X86 架构的 CISC 架构设计使其在执行复杂的动态内容处理任务时，能有效地管理和执行计算密集型操作，因而在 DLT 中展现出更高的性能。

综上所述，本研究不仅凸显了自动参数调优在提升 Nginx 服务器性能方面的重要性，还深入探讨了不同硬件架构上处理静态与动态负载测试时性能差异的原因。具体来说，本研究的主要发现如下：(1) 集成学习模型在 X86 和 ARM 硬件架构下均表现出高准确性，突显了跨平台性




(a) X86 架构在 SLT 与 DLT 中相对于默认配置的加速对比




(b) ARM 架构在 SLT 与 DLT 中相对于默认配置的加速对比

**图 5　X86 和 ARM 架构在静态负载测试(SLT)和动态负载测试(DLT)中相对于默认配置的加速对比**

**Fig. 5　X86 vs. ARM architecture acceleration comparison to default configuration in static load test (SLT) and dynamic load test (DLT)**

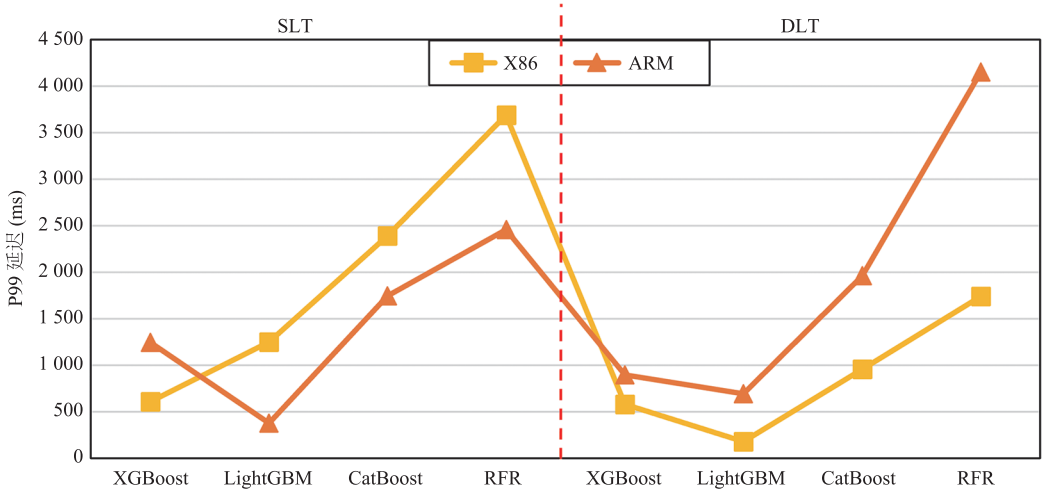



**图 6　X86 和 ARM 架构在静态负载测试(SLT)和动态负载测试(DLT)中的加速对比**

**Fig. 6　X86 vs. ARM architecture acceleration comparison in static load test (SLT) and dynamic load test (DLT)**

能优化中，模型选择的重要性；(2)在迭代次数上，ARM 架构优于 X86 架构，但在迭代收敛时间上，两种架构各有千秋，其中，ARM 在静态测试中更迅速，而 X86 在动态测试中更高效；(3)性能提升的对比分析揭示了 ARM 架构在处理静态内容时的优势和 X86 架构在动态内容处理上的高效性。这些发现为未来在多样化硬件平台上部署和优化 Web 服务器提供了理论与实践价值，强调了性能优化时考虑硬件架构特性的必要性，并为利用遗传算法进行操作系统参数优化的研究领域贡献了见解。

## 5 讨 论

本研究深入探讨了在 X86 和 ARM 硬件架构下进行 Nginx 性能调优的复杂性，揭示了其中的关键局限性和误差来源，并提出了不同硬件架构的特性及其迭代效率的考量。

### 5.1 本研究的局限性和误差来源

本节探究了 Nginx 性能调优在 X86 与 ARM 架构下的差异性及其背后的原因，揭示了优化过程中的关键局限性和潜在误差来源。

#### 5.1.1 局限性

(1)硬件架构的本质差异导致的性能波动：X86 和 ARM 架构的设计理念和实现方式不同，直接影响性能优化的适用性和效果。X86 架构的 CISC 架构设计优化了复杂操作的处理能力，适合执行多任务处理和高负载应用程序[6]。这一点在负载测试中得到了体现：X86 架构显示出比 ARM 架构更快的数据处理能力，尤其是在处理动态内容和高并发请求时；ARM 架构的 RISC 架构设计使其在能效和处理简单任务上具有优势，特别地，在能源消耗敏感的应用场景中表现更佳[7]。这些差异导致针对一种架构优化的策略不一定适用于另一种，增加了跨架构性能优化的复杂性。

(2)实验环境的限制性对泛化性的影响：本研究在固定的实验环境中进行，特别地，选用 Ubuntu 18.04.6 LTS 作为操作系统平台，可能限制了结果的泛化能力。操作系统版本之间的差异，如 Linux 内核的更新，可能带来网络栈、文件系统处理和安全性能的改进，从而影响 Nginx 的性能。由此表明，本研究的发现可能不完全适用于运行更新或不同版本 Linux 操作系统的环境，限制了本研究优化结果在不同技术背景下的应用范围。

(3)Nginx 配置多样性的考量不足：Nginx 性能的优化不仅取决于硬件架构和系统级参数，还受 Nginx 自身配置的影响[38]。在本研究的实验中，特定的参数设置(如 worker_processes 和 worker_connections)针对特定的应用场景进行了优化。然而，实际应用场景的多样性表明，这些配置可能并不具备广泛的适用性。例如，静态内容分发与动态内容生成对 Nginx 配置的需求存在显著差异。本研究未充分考虑这一点，可能限制了优化策略在不同应用场景中的适用性和效果，导致在某些场景下的性能提升效果不如预期。

通过深入分析上述局限性，本研究为未来的性能优化研究提供了参考，强调了在制定优化策略时全面考虑硬件架构特性、操作系统版本和 Nginx 配置多样性的重要性。

#### 5.1.2 误差来源

(1)网络条件的实际复杂性：在实验设置中，假定的理想网络环境忽略了带宽限制和丢包等关键因素，而这些因素在真实世界应用中对性能有决定性影响。例如，带宽限制和丢包现象可能会严重降低 Nginx 服务器处理高流量请求的能力[39]。因此，不考虑这些网络变量可能导致性能评估结果与实际应用场景存在偏差。

(2)性能评估模型的准确性问题：自动调优过程对性能评估模型的依赖性高，但这些模型的预测准确性受限于训练数据集的代表性和质量[40]。例如，若训练数据未覆盖 Nginx 在极端负载下的

行为，则模型可能无法准确预测这些情况下的性能表现。此外，模型超参数(如 XGBoost 的学习率和树的深度)的选择对预测结果有显著影响。不恰当的超参数设置可能导致模型过拟合或欠拟合，从而降低预测的准确性和可靠性。

(3)全局搜索策略的局限性：理论上，遗传算法虽然可以找到全局最优解，但在实际应用中，该算法的搜索效率和结果质量受种群大小、交叉率和变异率等算法参数的影响[41]。特别地，在参数空间复杂度高的情况下，遗传算法可能陷入局部最优解，无法达到真正的全局最优。此外，算法的收敛速度可能受参数维度的影响，导致在高维参数空间中寻找最优解需要长时间迭代。这些局限性表明，即使采用了先进的全局搜索策略，也无法保证总是能快速有效地找到最优的 Nginx 配置参数组合。

通过对这些误差来源的深入分析，本研究强调了在性能评估和优化过程中考虑实际网络条件、提高模型训练的质量和代表性，以及优化全局搜索策略的重要性，以确保 Nginx 性能调优研究的结果更具准确性和稳健性。

本研究的局限性和误差来源的深入剖析揭示了在 Nginx 性能优化研究中必须考虑的复杂性和挑战。本研究发现，硬件架构的本质差异、实验环境的限制性和 Nginx 配置的多样性均可能导致优化策略的适用性和有效性受限。此外，误差来源，如网络条件的实际复杂性、性能评估模型的准确性问题和全局搜索策略的局限性等，进一步增加了性能优化工作的难度。这些发现强调未来研究中需采取更全面的方法，如考虑实际网络环境、使用更精准的性能评估模型和改进全局搜索策略等，以提高性能优化的效率和效果。

### 5.2　不同硬件架构的特性及其迭代效率的考量

在探索不同硬件架构对 Nginx 性能优化的影响时，确定和利用架构特性、提高迭代效率的策略和综合考虑软硬件因素至关重要。

#### 5.2.1　确定和利用架构特性

(1)性能与能效的平衡：X86 架构适合高并发的 Web 服务，这一点在大型电商平台的实际部署中得到验证。其中，X86 处理并发请求的能力显著优于 ARM。相反，ARM 架构在物联网应用中展现出其能效优势，在降低能源消耗的同时，保持合理的响应速度，这对能源敏感的部署场景至关重要。

(2)针对特定硬件的优化策略：在静态内容分发中，ARM 架构因其较高的能效比显著降低了运营成本。实证表明，ARM 架构的能耗相比其他架构减少了约 40%[8]；而 X86 的多线程能力使其在动态内容生成中表现出色，例如，通过增加 Nginx worker 进程提升处理能力，这在处理社交媒体背景任务时特别有效。

(3)软件配置的硬件适配：根据 ARM 和 X86 的特点调整 Nginx 配置，如在 ARM 上降低 worker_processes，以减少能耗，而在 X86 上提高该值，以充分利用计算资源。通过这种方法，可在新闻网站等高流量场景中实现最佳性能。

#### 5.2.2　提高迭代效率的策略

(1)基于模型的参数调优：通过使用 XGBoost 等机器学习模型预测不同参数调整对性能的影响，提升调优的精度。与传统的盲目尝试相比，这种方法能使性能测试更加有针对性，可减少无效的迭代步骤，显著提高调优效率。

(2)并行测试与评估：利用 Docker 容器并行运行多个 Nginx 实例进行性能测试，大幅提高了测试效率。这种方法使得同时评估多种配置成为可能，缩短了优化周期。

(3)动态调整搜索策略：根据实时反馈调整遗传算法中的参数，如提高变异率，以探索更广的参数空间，针对性地优化对性能影响较大的参数，使搜索过程更加高效。

#### 5.2.3　综合考虑软硬件因素

(1)软硬件协同优化：Linux 内核参数和

Nginx 配置的调整需考虑硬件特性，如在 ARM 上优化 vm.swappiness，以减少交换操作，同时调整 Nginx worker_processes，以匹配硬件能力，实现软硬件的最佳匹配。

(2) 跨架构性能基准测试：定期执行基准测试，评估不同硬件架构的性能差异，如 ARM 处理静态请求的效率高于 X86，为系统管理员提供决策依据，同时监控由技术进步导致的性能变化。

(3) 灵活的优化策略：随技术进步和应用需求变化调整优化策略，如新兴 ARM 服务器的性能提升可能使其更适用于之前由 X86 主导的高性能计算任务。同时，优化策略应随云计算和大数据技术的演进而演变，以确保策略的现代性和适用性。

通过这些策略，本研究不仅能针对不同硬件架构的特性制定出更为精确和高效的性能优化措施，还能在快速变化的技术环境中保持优化策略的前瞻性和灵活性，为 Nginx 服务器的性能优化提供持续的支持。

综上所述，本研究深入探讨了 Nginx 性能调优过程中可能遇到的关键局限性和误差来源，并提出了针对不同硬件架构的优化策略及其迭代效率的考量。本研究强调了考虑硬件架构差异、实验设置限制和调优方法局限性的重要性，并探讨了提高迭代效率的策略。本研究旨在通过模型驱动的参数选择、并行测试评估和动态搜索策略调整提高性能优化的准确性和效率。此外，综合软硬件因素的考量对实现最佳性能调优也至关重要。本研究为系统管理员和开发者提供了在变化的技术环境中优化 Nginx 性能的实用指导，未来研究应继续改进策略，以满足日益增长的 Web 服务需求。

## 6 结 论

本研究通过对 X86 和 ARM 架构下 Nginx 服务器进行系统参数优化和性能测试，揭示了两种硬件架构在不同负载条件下的性能差异。结果表明，X86 架构在处理动态请求方面具有显著优势，而 ARM 架构则在静态请求处理上展现更高效能。这一发现强调了性能优化策略需针对硬件特性和负载类型进行定制。本研究还分析了 Nginx 性能提升的原因，包括硬件设计、资源调度效率和遗传算法迭代特性等因素。因此，为最大化 Nginx 服务器的性能，系统管理员需深入理解不同硬件架构的特点，以及如何根据这些特点调整系统参数，从而针对特定的负载条件实现优化。

## 参 考 文 献

[1] Nginx. Nginx news [EB/OL]. (2009-12-14)[2024-04-11]. https://nginx.org/.

[2] W3Techs. Usage statistics and market shares of Web servers [EB/OL]. (2024-02-01)[2024-04-11]. https://w3techs.com/technologies/overview/web_server/.

[3] Nguyen VN. A comparative performance evaluation of web servers [J]. VNU Journal of Science: Computer Science and Communication Engineering, 2017, 31(3): 28-34.

[4] Chen S, GalOn S, Delimitrou C, et al. Workload characterization of interactive cloud services on big and small server platforms [C] // Proceedings of the 2017 IEEE International Symposium on Workload Characterization, 2017: 125-134.

[5] Sankaralingam K, Menon J, Blem ER. A detailed analysis of contemporary arm and X86 architectures [R]. TR1783, Madison: University of Wisconsin-Madison, Department of Computer Sciences, 2013.

[6] TechSci Research LLC. X86 server market report [EB/OL]. (2023-10-23)[2024-04-11]. https://www.techsciresearch.com/report/x86-server-market/17414.html.

[7] Rahman TN, Khan N, Zaman ZI. Redefining computing: rise of ARM from consumer to cloud for energy efficiency [J]. World Journal of

Advanced Research and Reviews, 2024, 21(1): 817-835.

[8] Nginx. NGINX plus on arm-based AWS graviton2 AMIs [EB/OL]. (2024-03-03)[2024-04-11]. https://www.nginx.com/resources/datasheets/nginx-plus-arm-based-aws-graviton2-amis/.

[9] Jader OH, Zeebaree S, Zebari RR. A state of art survey for web server performance measurement and load balancing mechanisms [J]. International Journal of Scientific & Technology Research, 2019, 8(12): 535-543.

[10] Ma C, Chi YH. Evaluation test and improvement of load balancing algorithms of Nginx [J]. IEEE Access, 2022, 10: 14311-14324.

[11] Laghari A, He H, Laghari R, et al. Cache performance optimization of QoC framework [J]. EAI Endorsed Transactions on Scalable Information Systems, 2019, 6(20): e7.

[12] da Silva M, Husemann R, de Gomensoro Malheiros M. Comparação de desempenho das arquiteturas SOC de CPU ARM Cortex A57 e Intel X86 como servidores dos softwares Apache e Nginx [J]. Revista Destaques Acadêmicos, 2017, 9(4): 281-297.

[13] Kalyanasundaram J, Simmhan Y. ARM wrestling with Big Data: a study of commodity ARM64 server for Big Data workloads [C] // Proceedings of the 2017 IEEE 24th International Conference on High Performance Computing (HiPC), 2017: 203-212.

[14] Chen XH, Hung LH, Cordingly R, et al. X86 vs. ARM64: an investigation of factors influencing serverless performance [C] // Proceedings of the 9th International Workshop on Serverless Computing, 2023: 7-12.

[15] Jiang QY, Lee YC, Zomaya AY. The power of ARM64 in public clouds [C] // Proceedings of the 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), 2020: 459-468.

[16] Wang JD, Kai Z. Performance analysis and optimization of Nginx-based web server [J]. Journal of Physics: Conference Series, 2021, 1955(1): 012033.

[17] Sharma R. Nginx high performance [M]. Birmingham: Packt Publishing Ltd, 2015: 86-90.

[18] Wang RZ, Wang QL, Hu YX, et al. Industry practice of configuration auto-tuning for cloud applications and services [C] // Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2022: 1555-1565.

[19] Chen WX, Li LL, Yu ZB. INTOP: improving Nginx performance by tuning OS parameters [C] // Proceedings of the 2023 International Conference on High Performance Big Data and Intelligent Systems (HDIS), 2023: 16-23.

[20] DeJonghe D. Nginx cookbook [M]. Sebastopol: O'Reilly Media, 2020.

[21] Tene G, Barker M. wrk2: a constant throughput, correct latency recording variant of wrk [EB/OL]. (2019-09-22)[2024-04-11]. https://github. com/giltene/wrk2.

[22] 陶辉. 深入理解 Nginx: 模块开发与架构解析 [M]. 2 版. 北京: 机械工业出版社, 2016.
Tao Hui. Understanding Nginx: modules development and architecture resolving [M]. 2nd ed. Beijing: China Machine Press, 2016.

[23] Friedman JH. Greedy function approximation: a gradient boosting machine [J]. The Annals of Statistics, 2001, 29(5): 1189-1232.

[24] Çetin V, Yildiz O. A comprehensive review on data preprocessing techniques in data analysis [J]. Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi, 2022, 28(2): 299-312.

[25] Wang XZ, Zhao YX, Pourpanah F. Recent advances in deep learning [J]. International Journal of Machine Learning and Cybernetics, 2020, 11(4): 747-750.

[26] Brereton RG, Lloyd GR. Support vector machines for classification and regression [J]. Analyst, 2010, 135(2): 230-267.

[27] Segal MR. Machine learning benchmarks and random forest regression [R]. CA 94143-0560, San Francisco: University of California, San Francisco, 2003.

[28] Prokhorenkova L, Gusev G, Vorobev A, et al. CatBoost: unbiased boosting with categorical features [C] // Proceedings of the 32nd Conference on Neural Information Processing Systems, 2018: 6639-6649.

[29] Ke GL, Meng Q, Finley T, et al. LightGBM: a highly efficient gradient boosting decision tree [C] // Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017: 1-9.

[30] Chen TQ, Guestrin C. XGBoost: a scalable tree boosting system [C] // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016: 785-794.

[31] Raschka S. Model evaluation, model selection, and algorithm selection in machine learning [Z/OL]. arXiv Preprint, arXiv: 1811.12808, 2018.

[32] Wu J, Chen XY, Zhang H, et al. Hyperparameter optimization for machine learning models based on Bayesian optimization [J]. Journal of Electronic Science and Technology, 2019, 17(1): 26-40.

[33] Kumar M, Husian M, Upreti N, et al. Genetic algorithm: review and application [J]. International Journal of Information Technology and Knowledge Management, 2010, 2(2): 451-454.

[34] Guilmeau T, Chouzenoux E, Elvira V. Simulated annealing: a review and a new scheme [C] // Proceedings of the 2021 IEEE Statistical Signal Processing Workshop (SSP), 2021: 101-105.

[35] Wang DS, Tan DP, Liu L. Particle swarm optimization algorithm: an overview [J]. Soft Computing, 2018, 22(2): 387-408.

[36] Pant M, Zaheer H, Garcia-Hernandez L, et al. Differential evolution: a review of more than two decades of research [J]. Engineering Applications of Artificial Intelligence, 2020, 90: 103479.

[37] Sagi O, Rokach L. Ensemble learning: a survey [J]. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2018, 8(4): e1249.

[38] Aivaliotis D. Mastering Nginx [M]. Birmingham: Packt Publishing Ltd, 2016.

[39] Nedelcu C. Nginx HTTP server [M]. Birmingham: Packt Publishing Ltd, 2015.

[40] Cong G, Fan WF, Geerts F, et al. Improving data quality: consistency and accuracy [C] // Proceedings of the 33rd International Conference on Very Large Databases, 2007: 315-326.

[41] Sivanandam SN, Deepa SN. Applications of genetic algorithms [M]. Berlin: Springer Berlin Heidelberg, 2008.