第 9 卷 第 6 期
2020 年 11 月
集 成 技 术
JOURNAL OF INTEGRATION TECHNOLOGY
Vol. 9 No. 6
Nov. 2020

引文格式：
吴嘉澍, 王红博, 代浩, 等. 基于机器学习的动态分区并行文件系统性能优化 [J]. 集成技术, 2020, 9(6): 71-83.
Wu JS, Wang HB, Dai H, et al. Research on machine learning-based performance optimization of dynamic partitioned parallel file system [J]. Journal of Integration Technology, 2020, 9(6): 71-83.

基于机器学习的动态分区并行文件系统性能优化

吴嘉澍[1,2] 王红博[1] 代 浩[1] 须成忠[3] 王 洋[1]

[1]（中国科学院深圳先进技术研究院 深圳 518055）
[2]（墨尔本大学计算与信息系统学院 墨尔本 3052）
[3]（澳门大学物联网智慧城市国家重点实验室 澳门 999078）

摘 要 近年来，随着大数据、云计算技术的发展，应用系统越来越集中，规模亦越来越大，使得存储系统的性能问题越来越突出。为应对其性能要求，并行文件系统得到了大量的应用。然而现有的并行文件系统优化方法，大多只考虑应用系统或并行文件系统本身，较少考虑两者之间的协同。该文基于应用系统在并行文件系统上的访问模式对存储系统的性能有显著影响这一特点，提出基于动态分区的并行文件系统优化方法。首先，利用机器学习技术来分析挖掘各个性能影响因素和性能指标之间的关系和规律，生成优化模型。其次，以优化模型为基础，辅助并行文件系统的参数调优工作。最后，基于 Ceph 存储系统进行原型实现，并设计了三层架构应用系统进行了性能测试，最终达到优化并行文件系统访问性能的目的。实验结果表明，所提出方法可以达到 85% 的预测优化准确率；在所提出模型的辅助优化下，并行文件系统的吞吐量性能得到约 3.6 倍的提升。

关键词 并行文件系统；性能优化；机器学习；逻辑回归
中图分类号 TP 302 **文献标志码** A doi: 10.12146/j.issn.2095-3135.20200901001

Research on Machine Learning-Based Performance Optimization of Dynamic Partitioned Parallel File System

WU Jiashu[1,2] WANG Hongbo[1] DAI Hao[1] XU Chengzhong[3] WANG Yang[1]

[1](*Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen* 518055, *China*)
[2](*School of Computing and Information Systems, University of Melbourne, Parkville, Victoria* 3052, *Australia*)
[3](*The State Key Laboratory of IoT for Smart City, University of Macau, Macau* 999078, *China*)

Abstract Thanks to the rapid evolvement of technologies including big data and cloud computing,

收稿日期：2020-09-01 **修回日期**：2020-10-24
基金项目：广东省重点领域研发项目(2019B010137002, 2020B010164003)；国家自然科学基金面上项目(61672513)
作者简介：吴嘉澍，硕士研究生，研究方向为大数据与云计算；王红博，硕士，研究方向为云存储；代浩，博士研究生，研究方向为边缘人工智能；须成忠，教授，IEEE Fellow，研究方向为并行分布系统；王洋(通讯作者)，研究员，研究方向为云计算与并行分布式计算，E-mail：yang.wang1@siat.ac.cn。

application systems become more and more centralized with boosted scale, which gradually highlights the performance issues of storage systems. Parallel file systems have been applied in a wide range of applications to meet the performance requirements of large-scale applications running on the storage systems. However, the majority of currently used parallel file system optimization methods only takes the application system or the parallel file system itself into account, and seldom considers the collaboration among them. Considering that the access mode of an application system when accessing the parallel file system will have a significant impact on the storage system performance, this study proposes a parallel file system optimization approach based on dynamic partitioning. The key idea is to firstly leverage machine learning techniques to reveal the relationships between factors that can influence the system performance and build an optimization model accordingly. Then, the optimization model will facilitate the parameter optimization of parallel file systems. Finally, the model is tested on a Ceph-based storage system prototype with a three-layer application system. The proposed model successfully optimizes the parallel file system access performance. Experimentally, the proposed model achieves an optimization prediction accuracy of 85%. With the assistance of the proposed model, the system throughput is improved by 3.6 times.

Keywords parallel file system; performance optimization; machine learning; logistic regression

1 引 言

伴随着互联网、大数据和云计算技术的发展，应用系统的计算、数据和部署越来越集中，应用系统的规模也相应地不断扩大。对于输入输出(I/O)密集型的应用程序，存储系统性能，即其运行效率，常常会成为整个系统的性能瓶颈。常见的性能指标有吞吐量(Throughput)、每秒读写操作次数(Input/Output Operations per Second，IOPS)等。为了解决这一问题，研究者进行了各种各样的尝试，而并行文件系统便是一种被广泛应用的系统技术[1]。

并行文件系统通过将多个逻辑上独立的存储节点聚合为一个有逻辑的、高性能的存储系统来缓解或解决相应的存储系统性能问题。它可以将分散于各个存储节点上的磁盘性能进行汇聚，以统一的、标准的接口为应用提供高性能的文件存储及访问服务，并可以随着系统规模的扩大而很容易地进行纵向、横向的扩展，因而具有高吞吐量、高 I/O 带宽、容易扩展等特点[2]。

现实场景中的应用系统都具有多样性。应用系统本身的特点及其数据访问层的设计决定了其在并行文件系统上的访问需求及 I/O 访问模式。譬如对于视频监控类应用来说，其 I/O 访问模式主要以顺序读或写为主，强调并行文件系统的吞吐量；而股票交易系统的 I/O 访问模式则以随机读或写为主，强调 IOPS 的并发能力[3]。如何针对应用系统的 I/O 模式在并行文件系统的配置上进行优化，从而产生协同效果是一个值得思考的问题。而现有的性能优化方法，无论是基于并行文件系统本身，或是基于应用系统本身，都较少考虑这一点。

另一方面，当前提出的优化方法，绝大多数都是基于某一个时间点的、静态的优化方法[4-6]。当应用系统的功能、负载或者架构发生变化时，并行文件系统 I/O 访问模式也会随之产生变化，而这种变化将导致已有的优化方法可能不再适用于当前的系统。因此，根据某一个时间点应用系统的 I/O 模式对应用系统或并行文件系统进行孤立的、静态的优化方法都有其局限性。

在理想状况下，各种并行文件系统的设计假设所有的底层硬件都具有性能的一致性。正是出于这种假设，并行文件系统在设计时通常按照一定的算法将系统的 I/O 负载均匀地分配到每一个底层的数据存储节点上，但这种设计可能会存在底层存储系统的性能没有被充分利用的情况。因为在具体实践中，并行文件系统的各个存储服务器节点之间存在性能差异是常见的现象。不管这种差异是由于 I/O 控制方式造成的，还是由于其存储介质原理、运转机制等造成的，该差异在大多数情况下并没有在并行文件系统设计的时候被考虑，因而可能会造成性能较好的节点资源未被充分利用的情形。虽然这种情况可以通过应用数据存储分布策略和并发策略来弥补，但仍然值得留意。

综上所述，现有并行文件系统优化方法虽能在一定程度上缓解或解决性能问题，但是仍然存在不足。本文将寻求一种新的，更具有针对性、适应性的并行文件系统性能优化方法。本文的创新性及先进性主要体现在以下几点。

(1) 基于机器学习的动态分区并行文件系统框架：通过对各种应用 I/O 模式下影响并行文件系统性能的因素和性能指标数据进行相关性分析，发现并通过实验证实了块分区尺寸与两个主要性能度量指标之间的关系，据此再结合机器学习技术提出了基于机器学习的动态分区并行文件系统的优化方案。

(2) 基于逻辑回归的面向文件系统动态分区的机器学习算法：对基于动态分区的并行文件系统性能优化方法进行归纳和总结，提出基于逻辑回归的优化算法，从而实现通过输入调优参数后利用模型预测性能影响的方法来指导并行文件系统访问性能的参数调优工作。模型在验证数据集上达到了最大 85% 的准确率。

(3) 基于 Ceph 文件系统实现原型设计与实现：设计和实现了基于 Ceph 存储系统的并行文件系统原型，并用于机器学习模型所需数据的生成、模型的优化及验证工作。经过实验验证，整个环境可以满足本文所设计的各种实验及测试要求，可以用来验证所提出优化算法的可行性和有效性。

2　相关工作

近年来，并行文件系统被广泛应用，各种并行文件系统应运而生。比较著名的有 GPFS (IBM General Parallel File System)[7]、PVFS (Parallel Virtual File System)[8]、PanFS[9]、Lustre[10]及 Ceph[11]等。在进行并行文件系统的性能研究时，通常从研究其影响因素着手：寻找哪些因素对并行文件系统的性能产生影响并探索这些因素在应用系统、并行文件系统中的分布情况，进而研究各个因素对并行文件系统的影响程度。但是，由于并行文件系统是一个架构复杂的分布式系统，且需经常应对多变的应用系统而做出调整，因此造成了并行文件系统性能优化方法的复杂性。

并行文件系统性能优化研究通常从应用系统、元数据 (Metadata) 服务、存储服务器以及并行文件系统架构等几个方面展开 (见表 1)。在并行文件系统架构优化方面，可以采用的方法包括优化数据存储和分布的策略[4]、采用更高带宽的存储网络及采用基于负载特征的存储架构等[12]；对元数据优化的方法包括对元数据的创建、查找、搜索、存储分布进行分割优化[13]，其中对元数据服务器进行优化，采用分布式元数据服务架构等方法；对数据存储的优化方法包括优化数据存储条带化参数 (包括条带宽度、条带深度和带偏移量等)，优化磁盘类型和数量，RAID 级别，建立缓存机制，优化存储服务器数量[14]。

当需要在应用系统上着手对并行文件系统的访问性能进行优化时，通常可以从两方面着手：一是重构数据访问层，利用更优化的算法、采用异步消息队列、加入缓存机制或优化存储并行访

表 1　并行文件系统性能优化技术与方法

Table 1　Techniques and approaches for parallel file system performance optimization

架构优化	元数据 I/O 优化	数据 I/O 性能优化
基于负载特征的存储架构[1]	元数据创建优化[15]	优化存储条带化策略[16-18]
密集型元数据 I/O 缓存架构	元数据查找优化	I/O 负载均衡[7,19]
基于 Flash、DRAM 的存储系统架构	元数据搜索优化[20]	写操作优化
新型元数据管理架构[1]	元数据服务器优化	缓存技术[14]
分布式元数据架构	元数据存储优化[21-23]	客户端缓存技术[24-25]
基于 SSD、DRAM 和 HDD 的混合存储架构[4,26]		最小化访问冲突[27-28]
存储网络性能优化		最大化 I/O 并行访问能力[27]，带宽分配技术[29-33]

问策略等方法；二是从并行文件系统上着手，即本文所采取的方式，根据应用系统的特点和负载形式，对并行文件系统进行有针对性的配置从而达到优化性能的目的。

常见的并行文件系统性能优化方法见表 1，其对相关技术在包括架构、数据和元数据 I/O 性能等三方面进行了对比。

3　面向动态分区的并行文件系统性能优化算法

3.1　问题描述

并行文件系统的性能是指并行文件系统的运行效率。常见的性能评估指标有每秒读写操作次数 IOPS、吞吐量、最大每秒写 I/O 操作次数(WIOPS)和读 I/O 操作占比(RRatio)等。

通过对样本数据进行分析，构建热力图(图 1)并对其进行比较、分析之后，可以发现以下规律：

(1)区块尺寸(I/O 操作最大读、写分区尺寸)对两个主要性能 IOPS 和吞吐量的影响都极为显著。

(2)IOPS 随着区块尺寸的增加而减小，吞吐量则相反。进一步的分析发现，IOPS 及吞吐量和块分区尺寸近似线性相关。

(3)读 I/O 操作占比对 IOPS 和吞吐量也有影响，但没有块分区尺寸影响显著。

因此，本文选择动态分区尺寸大小作为改变性能的因素，对并行文件系统访问性能进行优化。本文选定通过调整区块尺寸(单次 I/O 操作最大读、写分区)来进行性能优化的另外一个原因是，现有的并行文件系统都支持对这一性能因素进行在线调整。

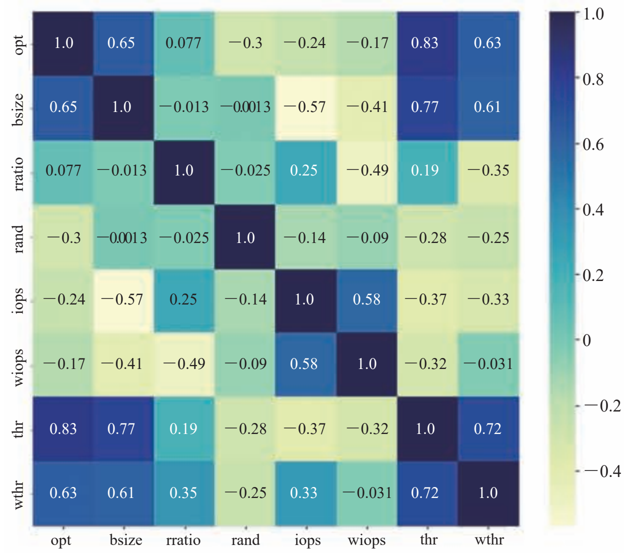


图 1　主要性能影响因素热力图

Fig. 1　Heatmap of major performance influencing factors

选定了性能因素之后，本文将利用机器学习的方法为块分区尺寸和并行文件系统性能指标建立关系模型，通过预测某个分区尺寸对并行文件系统性能的影响来指导性能优化工作，最终选取可对系统性能达到优化效果的分区尺寸并进行在线配置。

在性能优化评价方面，本文则选择 IOPS 和吞吐量来对并行文件系统访问性能的优化效果进

行评价。图 2 再次验证了分区尺寸对并行文件系统性能的显著影响，从而反映出通过调整分区尺寸对性能进行优化是可行的。为了对问题能更进一步的分析和研究，也为了更好地利用机器学习技术，本文需要做出一些假设和限定，并对所要研究的问题进行以下定义。

(1)假设与约束

并行文件系统硬件配置的更改，如使用固态硬盘(SSD)替代传统机械硬盘，会因其物理特性或技术特性的原因而显著地影响存储子系统的性能，因此本文假设并行文件系统的硬件配置在优化过程中保持不变。不仅如此，并行文件系统软件配置、架构的变更也可能对存储子系统性能产生影响，如缓存 Cache 的配置以及 Cache 的大小。因此本文也假设，在存储子系统优化过程中，除了本文选定的性能优化方法所需要的系统配置参数之外，其他系统配置、参数及系统架构在优化过程中均保持不变。

(2)性能优化目标

理想状态下，并行文件系统的最大存储访问性能(Capacity)由并行文件系统的物理特性和架构决定，并和具体的应用系统无关。因此在考虑优化目标时，将以并行文件系统的最大存储访问性能为基础并结合一定的性能损失来确定。存储系统的最大存储访问性能可以通过压力测试来获得，而应用系统对存储子系统的性能损失则依赖经验值，即 30%。在应用系统对并行文件系统访问性能损失参考值确定以后，就能以此来确定优化的目标。

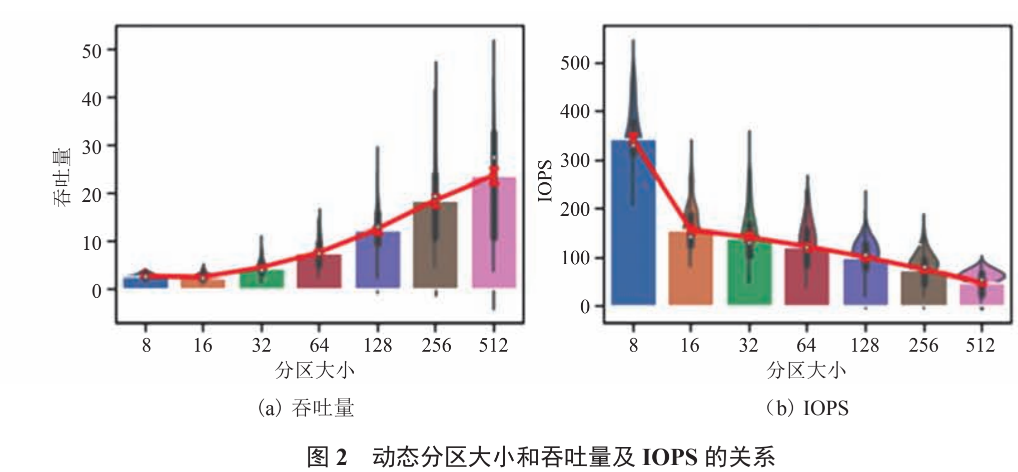
(3)问题的定义

依照本文的假设与约束，对所研究的问题进行了归纳，即针对特定应用系统特定的并行文件系统的访问模式，对其性能按照预先设定的优化目标，通过调整区块尺寸的方法进行优化。并且当应用系统对并行文件系统的访问模式发生变化时，所提出算法能够持续地调整优化方案，从而实现基于机器学习的动态并行文件系统的访问性能优化方法。

3.2　基于机器学习的多元逻辑回归算法

本文将问题转化为分类问题并采用逻辑回归算法来解决。逻辑回归是本文根据问题描述选择的机器学习算法，可以将并行文件系统性能优化的问题，转换成预测在并行文件系统上相关性能因素的修改能否达到预先设定的性能优化目标的二分类问题，并用于指导并行文件系统访问性能优化工作的参数调优。相较于传统依赖经验的性能参数调整方法，该方法更具有可操作性。

模型与算法的选择在任何机器学习项目中都极其关键，直接影响到后续预测结果的准确性和整个机器学习方法的适用性。好的模型选择不仅



(a) 吞吐量　　(b) IOPS

图 2　动态分区大小和吞吐量及 IOPS 的关系

Fig. 2　Relationships between dynamic block size and influencing factors (throughput and IOPS)

能提供较高的预测准确率，也理应具有良好的运算性能。本文所采用的逻辑回归算法计算量小、存储占用率低、实施简单高效且可以用于大数据量的场景[34]。除此之外，逻辑回归可以通过在线学习的方式更新参数，无需重新训练整个模型，这为实现动态调整性能影响因素，进而动态优化并行文件系统性能提供了帮助。

根据系统设计要求，多元逻辑回归的算法步骤如下：

(1) 运行 Jupyter 环境，加载 TensorFlow[35]、Pandas、Matplotlib 和 Numpy 模块。

(2) 在训练前先对数据进行随机排序处理并分配训练集和验证集，防止模型过拟合。其中，验证集包含 40 组数据。

(3) 读入准备好的训练和测试数据，用于后续训练及测试。

(4) 利用 Pandas 将读入的 csv 数据转换成二维数组。

(5) 采用预测函数：

$$h_\theta(x)=\text{sigmoid}(\boldsymbol{W}X+b) \tag{1}$$

其中，$\boldsymbol{W}$ 为权值矩阵；b 为偏置量 Bias。Sigmoid 激活函数输出并行文件系统能达到优化目标的概率，即

$$P(y=1|x,\theta) \tag{2}$$

损失函数采用交叉熵损失函数：

$$L(\theta)=-\frac{1}{m}\left\{\sum_{i=1}^{m}y^{(i)}\log\left[h_\theta(x^i)\right]+(1-y^{(i)})\log\left[1-h_\theta(x^i)\right]\right\} \tag{3}$$

其中，m 为样本数量。

(6) 定义两个变量并初始化用于存储 $\boldsymbol{W}$ 和 b。

(7) 选择 TensorFlow 的 GradientDescentOptimizer 进行梯度下降优化。设定优化目标，使得损失函数最小化。

(8) 在 Ceph 集群和应用服务器上进行训练和测试。训练过程中共进行 25 000 次训练。其中每 100 次为一批，共计 250 批次的训练。此种实验参数的选取足够使所训练的模型达到收敛状态。在每个批次的训练中，对当前训练批次的平均损失(Loss) 值和其在训练数据集上的平均准确率进行记录。最终通过观察和记录的损失值和准确率，了解和掌握整个训练过程。算法的代码节选如图 3 所示。

```
#构建预测函数
weight = tf.Variable(tf.random_normal([10,1]))
bias = tf.Variable(tf.random_normal([1]))
output = tf.matmul(x,weight) + bias
pred = tf.cast(tf.sigmoid(output) > 0.5, tf.float32)
#构建损失函数
loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits
(labels=y,logits=output))
# 梯度下降求 θ，X 为特征量，Y 为预测值
train_step = tf.train.GradientDescentOptimizer(0.0003).minimize(loss)
```

图 3　逻辑回归算法代码节选

Fig. 3　Code snippets of logistic regression algorithm

4　基于机器学习的动态分区并行文件系统优化方法

4.1　问题描述

通过第 3 小节对样本数据的分析，本文发现并行文件系统访问性能和动态分区尺寸之间存在近似线性的关系，同时在理论上讨论了如何运用机器学习算法对并行文件系统的访问性能进行优化。

为了实现所提出的机器学习算法，本文基于 Ceph 设计了相应的并行文件系统原型，并选择 IOMeter (www.iometer.org/ 2020) 作为 I/O 工具；在机器学习的框架和工具方面，则选用 Python 和 TensorFlow (https://www.tensorflow.org/. 2020) 对所提出算法进行实现。

4.2　基于 Ceph 的原型系统体系架构设计

4.2.1　系统原型设计相关技术

IOMeter 可以被用来测量磁盘和网络控制器

的性能、总线带宽和时延、驱动器的吞吐量、系统级别的硬件驱动性能等。整个工具由工作负载生成器 Dynamo 和 IOMeter 客户端组成。其中，Dynamo 可以按计划要求在相关应用系统中生成 I/O 负载，用于压力测试或 I/O 模拟。在具体测试中，Dynamo 将接受 IOMeter 发送来的指令并执行相应的 I/O 操作，同时记录各种性能指标信息并将数据返回给 IOMeter 生成测试报告。IOMeter 客户端则用于配置负载、设置操作参数、启动和停止测试，并在测试过程中收集测试结果数据。

IOMeter 具有很强的灵活性，可以很方便地利用脚本来定制 I/O 负载，并能检测系统的多种性能指标。这种灵活性使得它可以被配置用于模拟特定应用程序或基准测试程序的磁盘和网络 I/O，当然也可以用来产生综合性的 I/O 负载。

本文利用 IOMeter 对并行文件系统进行压力测试从而得到并行文件系统的性能。测试中模拟了各种 I/O 模式并记录性能相关信息，最后采用逻辑回归算法进行训练得出机器学习模型并用于性能优化参数效果的预测。

4.2.2 基于 Ceph 原型系统体系架构设计

本文利用 Ceph 平台实现了并行文件系统原型以进行相关的实验和测试，包括并行文件系统性能压力测试、机器学习样本数据生成、机器学习模型训练及验证等工作。应用系统采用常见的三层架构来实现，具体如图 4 所示。该三层架构分别为应用客户端、应用中间层和数据访问层，而并行文件系统构建则基于 Ceph 集群之上。其中，应用客户端主要用来进行运行自动化测试工具并监控和收集并行文件系统访问性能的相关数据；也可以用来运行相关数据分析、机器学习平台的工具。应用中间层则用来根据测试用例对特定的 I/O 访问模式进行模拟以生成相应的 I/O 负载，并将 I/O 的请求传递给数据层直至底层的基于 Ceph 的文件系统，其物理设计如见图 5 所示。

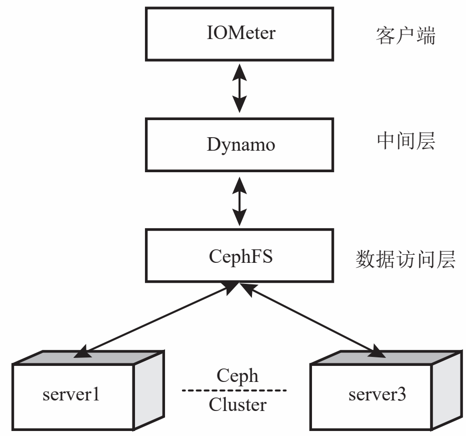


图 4　系统原型架构图

Fig. 4　System prototype structure

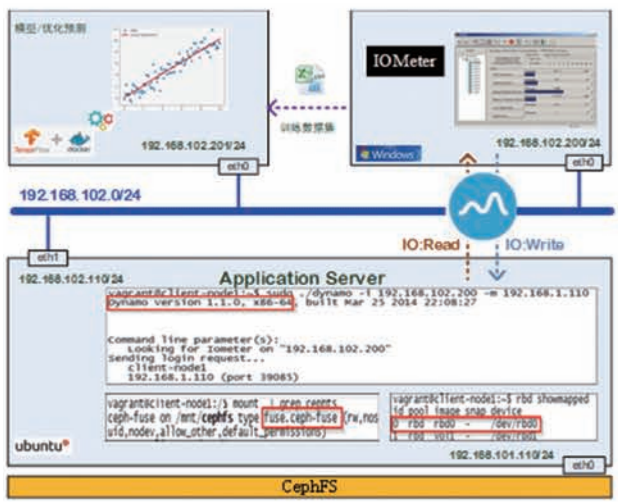


图 5　应用系统的物理设计

Fig. 5　Physical design of the application system

整个系统原型设计包括 7 台服务器，其中 5 台 Linux 服务器，2 台 Windows 服务器或工作站。在 Ceph 集群的存储节点上，采用了包括 SSD、机械硬盘和 SD 卡等各种存储介质的组合，以反映现实生产环境中存储介质的多样性，同时使得本研究更具鲁棒性。整个原型环境的配置如表 2 所示。

具体地，整个系统原型采用虚拟化的方式部署。在原型系统的搭建中，本文采用 Oracle VirtualBox[36]和 Vagrant[37]等相关的工具和技术。

其中，VirtualBox 是一个基于 x86 和 AMD64/Intel 的开源虚拟化软件包(Hypervisor)，在本实验中作为虚拟化主机来运行 Ceph 集群、应用服务器和测试服务器；而 Vagrant 是一个自动化管理工具，用于虚拟化环境的管理，如可以用来执行对虚拟机的配置、创建或运行进行自动化管理。文件系统的物理设计见图 6。

5 系统原型性能评测

5.1 原型系统部署

本研究创建了 3 个节点的 Ceph 集群来实现并行文件系统原型，同时使用 CentOS 6.0 和 Vagrant 2.2.4 启动 3 个虚拟机并检测虚拟机的状态，创建 Ceph 集群并配置 Ceph Monitor 服务，初始化现有硬盘并在上面创建对象存储器(OSD)，并基于 Ceph 文件系统来实现并行文件系统服务。Ceph 集群建立之后，再使用 1 台 Linux 的应用服务器和 1 台 Windows 机器来运行 IOMeter。

5.2 I/O 数据的收集与预处理

数据均来自于为本文所设计和部署的原型系统。为了避免数据的过拟合，也为了排除因为缓存等非必要因素对测试结果的影响，本文特别对测试时序和用例进行了精心设计，加入了预热时间。

表 2 原型机器环境配置

Table 2 Environment configuration of system prototype

功能	操作系统	硬件配置
IOMeter	Windows 7 SP1	Intel i7-2620M@2.7GHz, 4G RAM,1Gib LAN 40G SATA HDD
Application Server	Ubuntu 14.0.4	Intel i7-2620M@2.7GHz, 4G RAM,1Gib LAN 40G SATA HDD
Ceph Node#1	CentOS 6.0	Intel i7-2620M@2.7GHz, 1G RAM,1Gib LAN 20G SATA HDD * 3
Ceph Node#2	CentOS 6.0	Intel i7-2620M@2.7GHz, 1G RAM,1Gib LAN 20G SATA HDD *3
Ceph Node#3	CentOS 6.0	Intel i7-2620M@2.7GHz, 1G RAM,1Gib LAN 20G SATA HDD *3

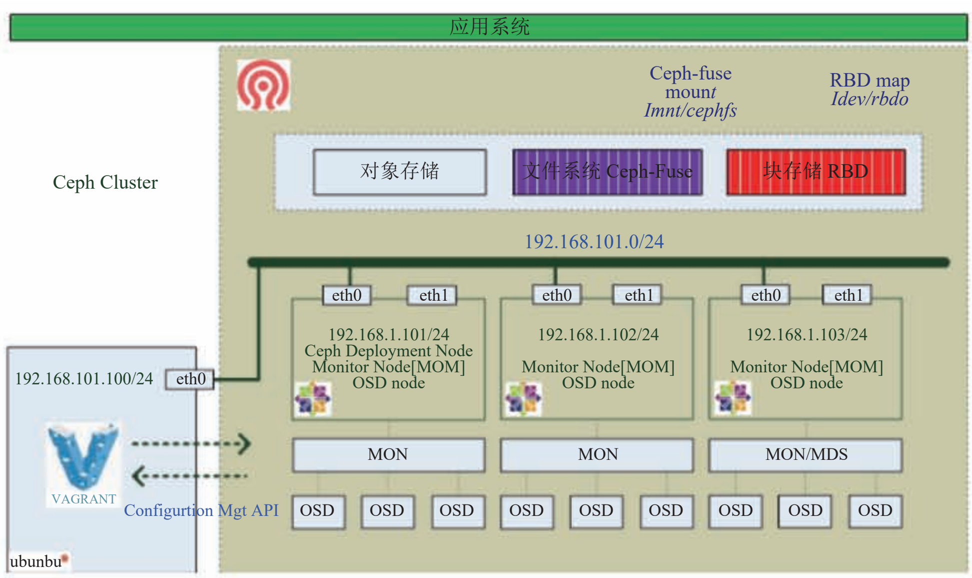


图 6 Ceph 集群物理设计图

Fig. 6 Physical design of Ceph cluster

在测试方法上，本文尽可能地列举了各种可能的 I/O 访问模式，并力求包括大多数常见 I/O 访问模式，然后根据每一种可能的 I/O 访问模式在并行文件系统上生成访问负载并记录相关的性能表现数据作为原始数据。在原型系统三层架构中，IOMeter 的 Dynamo 组件运行于客户端和中间层，用于生成 I/O 负载来模拟应用程序，同时收集性能数据。

对于优化指标，在选定吞吐量为优化指标时，优化目标根据最大吞吐量并考虑 30% 性能损失来确定。在训练数据集上，最大吞吐量为 43.828 MiBps，因此本文确定了优化目标为：

$$43.828\ \text{MiBps}\times(1-0.3)=30.680\ \text{MiBps}$$

即如果 I/O 分区模式的吞吐量大于或等于 30.680 MiBps 时，那么可以认为对应的应用系统在并行文件系统上的访问性能处于优化的状态；反之，则认为相关应用系统在相应的并行文件系统上的访问性能存在优化的空间，可以通过进一步修改分区尺寸的方式来进行优化。分区尺寸的选择可以参考图 2 揭示的 I/O 分区尺寸和吞吐量的关系。

在选定 IOPS 为优化指标时，优化目标则根据最大 IOPS 并考虑 30% 性能损失来确定。在训练数据集上，最大 IOPS 为 500，因此本文确定了优化目标为：

$$500\times(1-0.3)=350$$

即如果 I/O 分区模式的 IOPS 大于或等于 350 时，那么可以认为应用系统在并行文件系统上的访问性能处于优化的状态；反之，则认为相关应用系统在并行文件系统上的访问性能存在优化空间，可以通过进一步修改 I/O 尺寸的方式来进行优化。分区尺寸的选择同样可以参考图 2 揭示的 I/O 分区尺寸和 IOPS 的关系。

表 3 为清洗过的训练数据集节选，表头为本研究中选定的功能特征。以吞吐量为例，标记列的值表示某一行所描述的 I/O 模式的吞吐量是否已达到优化状态，其中 1 代表已达到优化状态，0 代表系统仍需优化。样本数据准备并标记好之后，将利用 Python 和 TensorFlow 来建立逻辑回归模型。

5.3　实验结果

5.3.1 验证过程及实验结果

在逻辑回归过程中，于训练数据集上进行了 25 000 次的训练，其中以每 100 次为一批进行训练。之后，采用随机分配的验证数据集对训练好的逻辑回归模型进行评估。对于验证集中的每一组数据，将模型预测的结果与真实标签进行比对，从而计算出模型准确率。

在本实验中，逻辑回归模型在训练数据集上的准确率可以达到 0.98，在测试数据集上的准确率则是 0.85，具体如表 4 所示。这表明，所训练的模型可以十分准确地预测出哪种尺寸设置能够

表 3　吞吐量优化训练数据集

Table 3　Training dataset of throughput performance optimization

标记	分区尺寸 (KiB)	读 I/O 百分比	随机 I/O	IOPS (MiBps)	IOPS 写 (MiBps)	吞吐量 (MiBps)	吞吐量写 (MiBps)
0	8	99	1	346.41	3.00	2.71	2.68
0	8	98	1	400.41	7.83	3.13	3.07
0	8	97	1	392.87	12.86	3.07	2.97
…	…	…	…	…	…	…	…
1	256	62	0	117.11	44.13	29.28	11.03
1	256	61	0	133.64	51.22	33.41	12.81
1	256	60	0	111.83	45.81	27.96	11.45
…	…	…	…	…	…	…	…

提高系统性能。同时，在经过约 100 轮训练后(图 7 和图 8)，训练趋于稳定、模型收敛，从而使得动态调整分区尺寸以及为不同系统设置不同分区尺寸参数变得更加可行。

表 4 逻辑回归训练过程中相关指标

Table 4 Evaluation metrics of logistic regression

训练阶段	训练指标		
	Loss 值	训练准确率	测试准确率
训练初期	41.06	0.52	0.55
训练后期	0.09	0.98	0.85

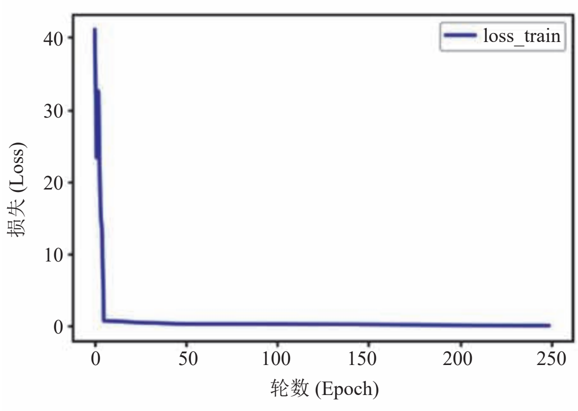


图 7 逻辑回归训练过程中的损失值

Fig. 7 Loss of logistic regression during training

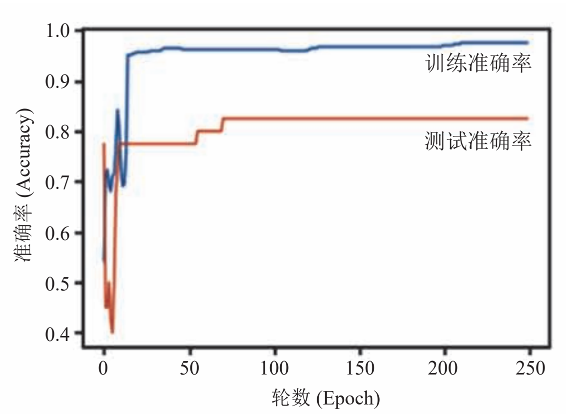


图 8 逻辑回归模型训练与测试准确率

Fig. 8 Logistic regression training and validation accuracy

5.3.2 结果实例分析

以优化吞吐量为例，在本文中，为了选取一个可以优化系统性能的分区尺寸，算法将进行以下过程：

(1) 假定首先选取 64 KiB 作为分区尺寸。如表 5 所示，实验得出的吞吐量 12.442 MiBps 小于性能优化目标 30.680 MiBps，则可以判定目前系统状态为非优化状态，需要进行优化。所以，此时的机器学习算法也应将此组数据分类为 0 类。本研究希望机器学习算法能够正确地判定此时的分区尺寸无法优化系统性能。

(2) 继续采用动态分区的方法调整 I/O 尺寸以进行优化。根据分区尺寸和吞吐量成近似线性关系，对于需要高吞吐量的系统而言，需要调高分区尺寸；对于需要高 IOPS 的系统而言，则需要降低分区尺寸。

(3) 调整分区尺寸为 128 KiB，测试系统的 I/O 模式和性能表现，如表 5 所示。此时系统性能未达到预期目标，所以本研究依旧希望机器学习模型也可将这组数据预测为 0 类，即经过模型预测，128 KiB 的分区尺寸对于当前系统状态而言不是一个好的选择，从而避免在此时选取 128 KiB 作为分区尺寸。

(4) 继续增大分区尺寸并进行预测，直至

表 5 优化案例 I/O 模式及性能数据

Table 5 I/O mode for cases of optimization and the corresponding performance

分区尺寸 (KiB)	读 I/O 百分比	随机 I/O	IOPS (MiBps)	IOPS 读 (MiBps)	IOPS 写 (MiBps)	吞吐量 (MiBps)	吞吐量读 (MiBps)	吞吐量写 (MiBps)
64	50	0	199.074	99.986	99.087	12.442	6.2491	6.192
128	50	0	87.403	43.435	43.969	10.925	5.429	5.496
256	50	0	92.377	44.971	47.406	23.094	11.243	11.852
512	50	0	87.657	43.678	43.978	43.828	21.839	21.989

512 KiB，预测评估当前系统是否处于优化状态。如表 5 所示，此时系统的吞吐量为 43.828 MiBps，大于预期优化目标 30.680 MiBps，此时系统处于优化状态。本研究期望机器学习模型可以将这组数据预测为 1 类，表示系统已处于优化状态，将分区尺寸设为 512 KiB 在此时是一个好的优化选择。

经过系统优化处理后，通过将分区尺寸从 64 KiB 调整到 512 KiB 后，应用系统并行文件系统访问性能从 12.442 MiBps 增加到 43.828 MiBps，增加了 3.6 倍，具体如图 9 所示。通过机器学习模型的帮助，当要动态调整分区尺寸时，就可以在模型的帮助下预测出可以提高系统性能的尺寸调整方案。

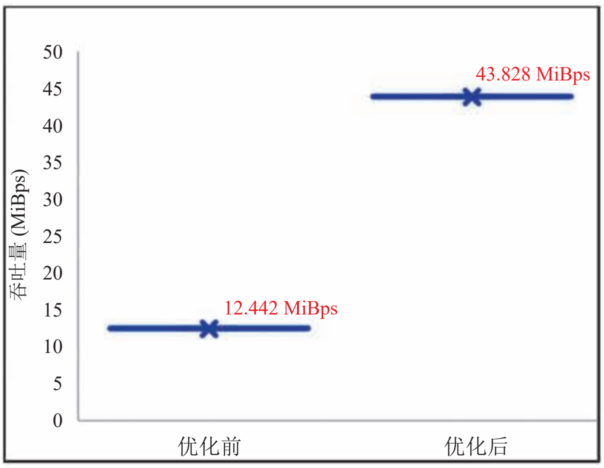


图 9　并行文件系统访问性能优化前后对比（吞吐量）

Fig. 9　Comparison of system throughput performance before and after configuration optimization

5.3.3 讨论与分析

从优化并行文件系统性能的框架上来看，相比于文献[13]，本文所提出的基于机器学习的动态分区并行文件系统优化架构可以自动地选取出合适的参数配置，从而达到对系统性能的优化，而文献[13]则需要通过实验尝试才能判断一个参数配置是否能够优化并行文件系统的性能。基于机器学习的优化框架通过其所学习到的信息来对参数配置进行判断，使得本方法更加灵活。

而从动态地对并行文件系统性能进行优化的角度来看，本文所提出的算法可以预先学习在不同系统 I/O 模式下的系统分区配置优化情况，从而使得动态地根据系统的实时 I/O 模式进行动态分区配置调整变得可行。而文献[13]所提出的方法则无法对系统性能进行动态的优化。

在优化所使用的算法上，本文采用了逻辑回归模型。随着深度神经网络的迅猛发展，在对并行文件系统进行优化的同时也可以考虑使用深度学习算法，如文献[14]中所使用到的深度强化学习算法，以学习更加灵活、鲁棒的优化模型。

6　总结与展望

本文基于应用系统的访问模式及分区尺寸对并行文件系统访问性能有显著影响这一特点，利用机器学习的技术对特定应用系统的 I/O 访问模式进行分析归纳，挖掘主要性能影响因素和主要性能指标之间的关系、规律并生成基于机器学习的优化模型，进而利用生成的机器学习模型来指导并行文件系统的参数优化工作，以达到优化并行文件系统整体访问性能的目的。

本文提出采用机器学习方法来进行并行文件系统的性能优化，但具体实现时，仅以分区尺寸这一主要的性能因素进行；评价性能也只选用两个最常用的性能评价指标 IOPS 和吞吐量，因此未来可以进一步地探索其他的性能影响因素和性能评价指标。

参 考 文 献

[1]　肖利民, 霍志胜. 大数据存储系统 I/O 性能优化技术研究进展 [J]. 大数据, 2017, 3(6): 65-84.

[2]　肖伟, 赵以爽. 并行文件系统简介及主流产品对比 [J]. 邮电设计技术, 2012(7): 31-36.

[3]　Yadgar G, Gabel M. Avoiding the streetlight effect: I/O workload analysis with SSDs in mind [C] // The

8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16), 2016.

[4] He SB, Wang Y, Sun XH, et al. HARL: optimizing parallel file systems with heterogeneity-aware region-level data layout [J]. IEEE Transactions on Computers, 2017, 66(6): 1048-1060.

[5] He SB, Wang Y, Li Z, et al. Cost-aware region-level data placement in multi-tiered parallel I/O systems [J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(7): 1853-1865.

[6] He SB, Liu Y, Wang Y, et al. Enhancing hybrid parallel file system through performance and space-aware data layout [J]. The International Journal of High Performance Computing Applications, 2016, 30(4): 396-410.

[7] Schmuck FB, Haskin RL. GPFS: a shared-disk file system for large computing clusters [C] // Proceedings of the Conference on File and Storage Technologies (FAST'02), 2002: 231-244.

[8] PVFS. Parallel virtual file system, sersion 2 [EB/OL]. 2016[2020-10-23]. https://web.archive.org/web/20160701052501.

[9] PANASAS. PanFS: the most adaptable parallel file system [EB/OL]. 2011[2020-10-23]. https://www.panasas. com/panfs-architecture/panfs/.

[10] Lustre. Lustre filesystem [EB/OL]. 2013[2020-10-23]. https://lustre.org/.

[11] Ceph. Ceph documentation [EB/OL]. 2020[2020-10-23]. https://docs.ceph.com/en/latest/.

[12] 赵铁柱. 分布式文件系统性能建模及应用研究 [D]. 广州: 华南理工大学, 2011.

[13] Jeong K, Duffy C, Kim J, et al. Optimizing the Ceph distributed file system for high performance computing [C] // 2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2019: 446-451.

[14] Wang YM, Li YB, Lan T, et al. DeepChunk: deep Q-learning for Chunk-based caching in wireless data processing networks [J]. IEEE Transactions on Cognitive Communications and Networking, 2019, 5(4): 1034-1045.

[15] Zhang ZH, Ghose K. hFS: a hybrid file system prototype for improving small file and metadata performance [C] // Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, 2007: 175-187.

[16] Dong B, Li XQ, Xiao LM, et al. A new file-specific stripe size selection method for highly concurrent data access [C] // Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing, 2012, DOI: 10.1109/Grid.2012.11.

[17] Lucas HV, Ludwig T. Hint controlled distribution with parallel file systems [C] // European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting, 2005: 110-118.

[18] Scheuermann P, Weikum G, Zabback P. Data partitioning and load balancing in parallel disk systems [J]. The VLDB Journal—The International Journal on Very Large Data Bases, 1998, 7(1): 48-66.

[19] Carns PH, Ligon Ⅲ WB, Ross RB, et al. PVFS: a parallel file system for Linux clusters [C] // Proceedings of the 4th Annual Linux Showcase and Conference, 2000: 317-328.

[20] Huo ZS, Xiao LM, Zhong QL, et al. MBFS: a parallel metadata search method based on Bloomfilters using MapReduce for large-scale file systems [J]. The Journal of Supercomputing, 2016, 72(8): 3006-3032.

[21] Johnson C, Keeton K, Morrey Ⅲ CB, et al. From research to practice: experiences engineering a production metadata database for a scale out file system [C] // Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST'14), 2014.

[22] Niazi S, Ismail M, Haridi S, et al. HopsFS: scaling hierarchical file system metadata using newSQL databases [C] // The 15th USENIX Conference on File and Storage Technologies (FAST'17), 2017: 89-104.

[23] Zhang SL, Catanese H, Wang AIA. The composite-file file system: decoupling the one-to-one mapping of files and metadata for better performance [C] // FAST'16: Proceedings of the 14th Usenix Conference on File and Storage Technologies, 2016: 15-22.

[24] Vilayannur M, Nath P, Sivasubramaniam A.

Providing tunable consistency for a parallel file store [C] // FAST’05: 4th USENIX Conference on File and Storage Technologies, 2005: 17-30.

[25] Vilayannur M, Sivasubramaniam A, Kandemir M, et al. Discretionary caching for I/O on clusters [C] // Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003.

[26] Wu HJ, Zhu LM, Lu K, et al. StageFS: a parallel file system optimizing metadata performance for SSD based clusters [C] // 2016 IEEE Trustcom/BigDataSE/ISPA, 2016: 2147-2152.

[27] Chen PM, Patterson DA. Maximizing performance in a striped disk array [C] // The 17th Annual International Symposium on Computer Architecture, 1990, DOI: 10.1109/ISCA.1990.134542.

[28] Verma A, Anand A. General store placement for response time minimization in parallel disks [J]. Journal of Parallel and Distributed Computing, 2007, 67(12): 1286-1300.

[29] Demers A, Keshav S, Shenker S. Analysis and simulation of a fair queueing algorithm [C] // ACM SIGCOMM Computer Communication Review, 1989, 19(4): 1-12.

[30] Ghodsi A, Sekar V, Zaharia M, et al. Multi-resource fair queueing for packet processing [J]. ACM SIGCOMM Computer Communication Review, 2012, 42(4): 1-12.

[31] Gulati A, Shanmuganathan G, Zhang XC, et al. Demand based hierarchical QoS using storage resource pools [C] // Proceedings of the 2012 USENIX Annual Technical Conference, 2012.

[32] Kim H, Seshadri S, Dickey CL, et al. Evaluating phase change memory for enterprise storage systems: a study of caching and tiering approaches [C] // Proceedings of the 12th (USENIX) Conference on File and Storage Technologies (FAST’14), 2014.

[33] Psomas CA, Schwartz J. Beyond beyond dominant resource fairness: indivisible resource allocation in clusters [R]. Tech Report Berkeley, 2013.

[34] Maalouf M. Logistic regression in data analysis: an overview [J]. International Journal of Data Analysis Techniques & Strategies, 2011, 3(3): 281-299.

[35] Abadi M, Barham P, Chen JM, et al. Tensorflow: a system for large-scale machine learning [C] // Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, 2016: 265-283.

[36] Oracel. Oracle VM virtualbox [EB/OL]. 2014[2020-20-23]. https://www.virtualbox.org/.

[37] HashiCrop. Vagrantup by HashiCorp [EB/OL]. 2013[2020-20-23]. https://www.vagrantup.com/.