基于强化学习的 Lustre 文件系统的性能调优

张文韬[1,2] 汪璐[1] 程耀东[1]
[1]中国科学院高能物理研究所计算中心，北京 100049
[2]中国科学院大学，北京 100049
(zhangwt@ihep.ac.cn)

Performance Optimization of Lustre File System Based on Reinforcement Learning

Zhang Wentao[1,2] Wang Lu[1] Cheng Yaodong[1]
[1](Computing Center, Institute of High Energy Physics, Chinese Academy of Sciences, Beijing 100049)
[2](University of Chinese Academy of Sciences, Beijing 100049)

Abstract Computing of high energy physics computing is a typical data-intensive application. The file access mode which is mainly skipping has high concurrency. The throughput and response time is very important to the whole system, and they are often the significant target of performance optimization. There are a large number of parameters that can be adjusted in a distributed storage system. The setting of these parameters has a great influence on the performance of the system. However, there is a delay between the parameter adjustment and the feedback from the system, if you take multiple consecutive adjustment actions, it is difficult to determine which action has played a role, or how much each action has affected the result. As a result, manual adjustments may be incorrect. Furthermore, the large parameter search space, load continuity and device diversity also determine that traditional methods are very inefficient. In fact, if the tuning engine is regarded as an agent and the storage system is regarded as the environment, the parameter adjustment problem of the storage system is a typical sequential decision problem. Therefore, based on data access characteristics of high energy physics calculation, we propose an automated parameter tuning method using the reinforcement learning. Experiments show that in the same test environment, using the default parameters of the Lustre file system as a benchmark, this method can increase the throughput by about 30%.

Key words distributed storage; reinforcement learning; performance tuning; deep learning

摘要 高能物理计算是典型的数据密集型计算，文件访问模式以跳读为主，并发度高。吞吐率、响应时间等性能对整个系统至关重要，往往是重点关注的性能优化目标。分布式存储系统存在大量可供调节的参数，这些参数的设置对系统的性能有着很大的影响。而参数调节和系统的反馈之间是有延时的，如果采取了连续多个调节动作，很难确定究竟是哪个动作起了作用，或者每个动作对结果的影响是多少。因此，人工调节不免存在偏差，况且庞大的参数搜索空间、负载的连续性、负载和设备的多样性等因素也决定了传统方法是非常低效的。实际上，如果把调节引擎看作是智能体，把存储系统看作是环境，存储系统的参数调节问题是典型的顺序决策问题。因此，基于高能物理计算的数据访问特点，我们提出了用强化学习的方法来进行自动化的参数调优。实验表明，在相同的测试环境下，以 Lustre 文件系统默认参数为基准，该方法可使其吞吐率提升 30%左右。

关键词 分布式存储；强化学习；性能调优；深度学习

基金项目：本文受国家重点研发计划项目：面向高能物理领域科学发现的高性能应用软件系统研制（2017YFB0203203）和国家自然科学基金项目：高能物理实验的大规模离线数据存储技术研究（11575223）资助。

1 背景

高能物理计算是一种数据密集型计算，存储系统是其性能决定因素之一。未来 10 年内，各大物理实验如江门中微子实验（JUNO）、高海拔宇宙线实验（LHAASO）、北京谱仪实验（BESIII）等将累积近 100 PB 的物理数据。大规模的数据处理给存储系统提出了“百 GB/s 的聚合带宽、数万个客户端并发访问、数据可靠性和可用性、跨域站点数据共享以及数据长期保存”等需求和挑战。

高能物理数据处理主要包括模拟计算、重建计算以及物理分析 3 种类型，每种计算类型各有其特点。高能物理计算中数据是一次写入，多次读取，通过监控计算节点，得到文件系统的访问模式，即大部分文件的连续读请求大小分布在 256K 到 4M 之间，每两个连续读请求之间都有 offset，65%的 offset 绝对值分布在 1M~4M 之间，这说明文件的读访问方式为大记录块的跳读。

存储系统管理员往往会根据系统的历史情况以及系统的实时状态，调节相应的参数值以提高系统的性能[5]。参数调节和系统的反馈之间是有延时的，如果采取了连续多个调节动作，很难确定究竟是哪个动作起了作用，或者每个动作对结果的影响是多少。因此，人工调节不免存在偏差，况且庞大的参数搜索空间、负载的连续性、负载和设备的多样性等因素也决定了传统方法是非常低效的。传统的参数配置算法包括基于模型的控制反馈算法和无模型的参数搜索两大类。前者需要系统管理员具备丰富的先验知识且不支持动态负载变化，后者在参数搜索空间很大时，优化效率很低。

强化学习由于其优秀的决策能力在人工智能领域得到了广泛应用。然而，早期的强化学习主要依赖于人工提取特征，难以处理复杂高维状态空间下的问题。随着深度学习的发展，算法可以直接从原始的高维数据中提取出特征。深度学习具有较强的感知能力，但是缺乏一定的决策能力；而强化学习具有较强的决策能力，但对感知问题束手无策。因此，将两者结合起来，优势互补，能够为复杂状态下的感知决策问题提供解决思路。

把调节引擎看作是智能体，把存储系统看作是环境，存储系统的参数调节问题是典型的顺序决策问题。因此，我们很自然地将强化学习引入到存储系统的参数调节中。本文基于高能物理计算的数据访问特点，设计并实现了基于强化学习的参数调节系统，实验表明，该系统可使 Lustre 文件系统的吞吐率提升 30%左右。

2 相关工作

2.1 强化学习基础

马尔可夫决策过程（Markov Decision Process，MDP）是强化学习的最基本的理论模型[13]。

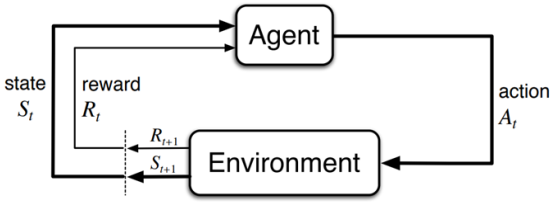

图 1 强化学习交互过程

Fig.1 Interactive process

在 MDP 中，Agent 和环境之间的交互过程可如图 1 所示。Agent 的目标是找到一个最优策略π^*，使得它在任意状态 s 和任意时间步骤 t 下，都能够获得最大的长期累积奖赏，即：

$$\pi^* = argmax_\pi E_\pi[\sum_{k=0}^{\infty}\gamma^k r_{t+k}|s_t = s] \quad (1)$$

强化学习的目标是寻找最优状态值函数（Optimal State Value Function）：

$$V^*(s) = max_\pi E_\pi[\sum_{k=0}^{\infty}\gamma^k r_{t+k}|s_t = s] \quad (2)$$

基于此推导出贝尔曼方程：

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma V_\pi(s')] \quad (3)$$

强化学习的方法分基于模型的方法和无模型的方法，而现实世界中大部分问题模型是未知的，所以解决无模型的方法是强化学习的精髓。无模型即意味着状态转移概率是未知的，这样计算值函数时期望是无法计算的，如何计算期望呢？强化学习在此处引入了蒙特卡罗方法。

蒙特卡罗法，即经验平均法。所谓经验，是指利用该策略做很多次试验，产生很多幕数据，这里一幕是一次试验的意思。平均就是求均值。利用蒙特卡罗方法求状态 s 处的值函数时，又可以分为第一次访问蒙特卡罗方法和每次访问蒙特卡罗方法。计算公式为：

$$V_\pi(s) = \frac{G_{11}(s)+G_{12}(s)+\cdots}{N(s)} \quad (4)$$

蒙特卡罗的方法需要等到每次试验结束，所以学习效率不高，于是人们又提出了时间差分法(Temporal-Difference)，其是蒙特卡罗法与动态规划法的结合，不用等到实验结束，而是在每步都更新。TD 方法更新值函数的公式为：

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \quad (5)$$

根据行动策略和评估策略是否是一个策略，时间

差分法又分同策略法（on-policy）与异策略法（off-policy）。时间差分法的同策略法即 Sarsa 方法，异策略法即 Q-learning 方法。

2. 2 强化学习算法

本节介绍调节系统用到的 3 种强化学习算法：DQN，A2C，PPO。

2.2.1 DQN

DQN 是基于 Q-learning 的算法，其对 Q-learning 的修改主要体现在以下三个方面[10]：

（1）DQN 利用深度卷积神经网络逼近值函数。利用神经网络逼近值函数的做法在强化学习领域早就存在了，可以追溯到上个世界 90 年代。当时人们发现用深度神经网络去逼近值函数常常出现不稳定不收敛的情况，所以这个方向一直没有突破，那 DQN 做了什么其他的事情呢？

（2）DQN 利用了经验回放对强化学习的学习过程进行训练。人在睡觉的时候，海马体会把一天的记忆重放给大脑皮层。利用这个启发机制，DQN 构造了一种新的神经网络训练方法：经验回放。对神经网络进行训练时，存在的假设是独立同分布。而通过强化学习采集到的数据之间存在着关联性，利用这些数据进行顺序训练，神经网络当然不稳定。经验回放可以打破数据间的关联，从而使网络得以收敛。

（3）DQN 独立设置了目标网络来单独处理时间差分算法中的 TD 偏差。我们称计算 TD 目标时所用的网络为 TD 网络。以往的神经网络逼近值函数时，计算 TD 目标的动作值函数所用的网络参数θ，与梯度计算中要逼近的值函数所用的网络参数相同，这样就容易使得数据间存在关联性，训练不稳定。为了解决这个问题，DQN 中计算 TD 目标的网络表示为θ^-，计算值函数逼近的网络表示为θ，用于动作值函数逼近的网络每一步都更新，而用于计算 TD 目标的网络每个固定的步数更新一次。因此值函数的更新变为：

$$\theta_{t+1} \leftarrow \theta_t + \alpha[r + \gamma max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)]\nabla Q(s, a; \theta) \quad (6)$$

2.2.2 从 AC 到 A2C

当要解决的问题动作空间很大或者动作为连续集时，值函数方法无法有效求解。策略梯度法是将策略进行参数化为$\pi_\theta(s)$，利用线性或非线性函数对策略进行表示，此时强化学习的目标回报函数可表示为

$$U(\theta) = \sum_\tau P(\tau; \theta) R(\tau) \quad (7)$$

τ表示智能体的行动轨迹。我们的训练方法则可以表示为

$$\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta U(\theta) \quad (8)$$

此时问题的关键是如何计算策略梯度$\nabla_\theta U(\theta)$

$$\nabla_\theta U(\theta) = \sum_\tau P(\tau; \theta) \nabla_\theta log P(\tau; \theta) R(\tau) \quad (9)$$

利用经验平均来估算梯度

$$\nabla_\theta U(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} (\sum_{t=0}^{H} \nabla_\theta log \pi(a_t | s_t; \theta) R_t) \quad (10)$$

该公式的意义在于，回报越高的动作越努力提高它出现的概率。但是某些情形下，每个动作的总回报R_t都不为负，那么所有的梯度值都大于等于 0，此时每个动作出现的概率都会提高，这在很大程度下减缓了学习的速度，而且也会使梯度的方差很大。因此需要对R_t使用某种标准化操作来降低梯度的方差。具体地，可以让R_t减去一个基线 b（baseline），b 通常设为R_t的一个期望估计，通过求梯度更新θ，总回报超过基线的动作的概率会提高，反之则降低，同时还可以降低梯度方差（证明略）。这种方式被叫做行动者-评论家（actor-critic）体系结构。

A3C（Asynchronous Advantage Actor Critic）算法为了提升训练速度采用异步训练的思想，利用多个线程[9]。每个线程相当于一个智能体在随机探索，多个智能体共同探索，并行计算策略梯度，对参数进行更新。相比 DQN 算法，A3C 算法不需要使用经验池来存储历史样本并随机抽取训练来打乱数据相关性，节约了存储空间，并且采用异步训练，大大加倍了数据的采样速度，也因此提升了训练速度。与此同时，采用多个不同训练环境采集样本，样本的分布更加均匀，更有利于神经网络的训练。

在 A3C 的基础上，OpenAI 又提出了 A2C（Synchronous Advantage Actor Critic）。如图 2 所示，两个算法的不同点在于，在 A3C 中，每个智能体并行独立地更新全局网络，因此，在特定时间，智能体使用的网络权重与其他智能体是不同的，这样导致每个智能体使用不同的策略在探索更多的环境。而在 A2C 中，所有并行智能体的更新先被统一收集起来，然后去更新全局网络，全局网络更新完后再将权重分发到各个智能体。为了鼓励探索，每个智能体最后执行的动作会被加入随机噪声。

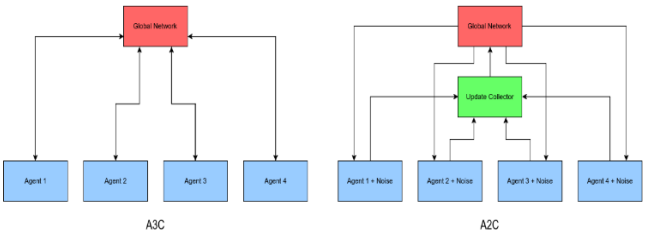

图 2 A3C 与 A2C

Fig.2 A3C VS A2C

2.2.3 从 TRPO 到 PPO

对于普通的策略梯度方法，如果更新步长太大，则容易发散；如果更新步长太小，即使收敛，收敛速

度也很慢，因而实际情况下训练常处于振荡不稳定的状态。Shulman 等为了解决普通的策略梯度算法无法保证性能单调非递减而提出了 TRPO（trust region policy optimization）算法[11]，TRPO 的主要期望是，我可以设置较大的步长，让学习快点，同时对目标函数进行优化时，有一定的约束条件，满足该约束条件后，优化是安全的，并能从数学上证明优化单调递增。约束表示新旧策略差异的 KL 散度期望小于一定值情形下，最大化下面目标表达式，通过 KL 散度来表示新旧策略差异，它小于一定值表示一个置信域，在这个置信域内进行优化。其目标函数如下：

$$max_\theta E\left[\frac{\pi_\theta(s,a)}{\pi_{\theta'}(s,a)}A_{\theta'}(s,a)\right],$$

$$\text{s.t. } E[D_{KL}(\pi_{\theta'}(s,\cdot)||\pi_\theta(s,\cdot))]\leq \delta. \quad (11)$$

TRPO 的标准解法是将目标函数进行一阶近似，约束条件利用泰勒进行二阶展开，然后利用共轭梯度的方法求解最优的更新参数。然而当策略选用深层神经网络表示时，TRPO 的标准解法计算量会非常大。因为共轭梯度法需要将约束条件进行二阶展开，二阶矩阵的计算量非常大。PPO 是 TRPO 的一阶近似[12]，把 TRPO 中的约束放到目标函数中，减少了计算量，可以应用到大规模的策略更新中，其目标函数如下：

$$J_{ppo}(\theta)=\sum_{t=1}^{T}\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta'}(a_t|s_t)}A_{\theta'}(a_t|s_t)-\beta\mathrm{KL}[\pi_{\theta'},\pi_\theta] \quad (12)$$

2.3 参数调优

参数调优是一个充满挑战的研究领域。

传统的方法有控制反馈算法与参数搜索算法。控制反馈算法是基于模型的方法，当已知负载运行情况且其简单时效果良好，但是此方法往往需要管理员设置关键参数的取值[3]。参数搜索算法往往是针对特定系统的特定负载进行一次搜索的过程[6]，当负载变化时其不适用。

张等提出了运用神经网络来加速传统的搜索方法[14]；陈等最先尝试了运用深度强化学习算法来进行参数调优，但是只是针对一台单独的服务器[2]；李等开发了 CAPES 来对更大规模的集群进行调参[7]，但是其存在以下缺点：

(1) 调节参数的范围较小，其只调节了 2 个参数。
(2) Reward 的设置过于简单，不能适应复杂且动态变化的负载。
(3) 状态 state 的设置有误，其将每个客户端的状态信息作为智能体的输入，但是智能体针对每个客户端的决策应是相互独立的，即智能体的输入应是每个单独客户端的状态，返回的动作信息应由接口负责将其正确分发。
(4) 训练过程未做批标准化，这样会导致训练不稳定，模型难以收敛。
(5) 强化学习算法发展迅速，最初的 DQN 算法无论是在训练速度以及模型效果上，已经与前沿算法有了显著差距。

3 设计与实现

如图 3 所示，系统由策略节点和目标集群组成。策略节点包含强化学习智能体以及信息接口，目标集群上的每个节点都包括一个用来收集节点信息的 monitor 和一个用来执行参数调节动作的 actor。系统运行时，每个节点的 monitor 每隔固定的时间会收集系统状态信息，并将信息发送给接口，接口把状态信息发送给智能体，智能体根据状态信息返回动作信息，并将其发回给接口，由接口将该动作发送给对应的节点，该节点的 actor 负责执行调节动作。然后不断迭代执行上述过程，直至满足终止条件。在这整个的交互过程中，强化学习智能体是在不断训练提升的。

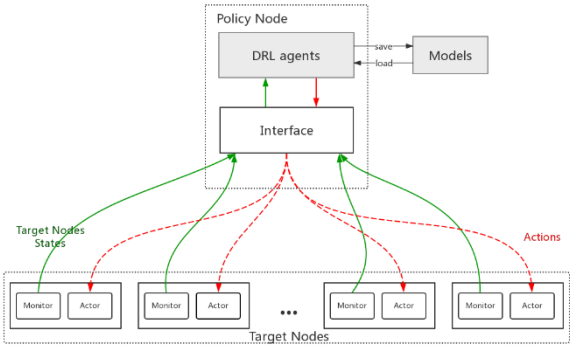

图 3 系统框架

Fig.3 system framework

在实际的部署过程中，策略节点应部署在目标集群外，以尽量降低系统运行过程中对目标集群的影响。并且，策略节点应尽可能地部署在含有 GPU 的节点上，以加快训练速度。

3.1 状态

类似于人类需要对环境信息了若指掌后才能做出好的决策一样，强化学习中的状态信息是十分重要的，算法会分析这些信息并做出决策。传统的机器学习方法需要大量的特征工程，近年来深度学习的飞速发展，已经证明其拥有强大的感知能力，所以深度强化学习算法只需要把状态信息输入深度神经网络，神经网络会自动抽取重要特征进行训练。

在做状态信息选取时，应尽可能地把跟系统性能相关的因素都包含进来[8]。本文选取的部分状态信息如下：

(1) Read throughput

读吞吐率

(2) Write throughput
写吞吐率

(3) dirty_pages_hits
脏页缓存命中的写操作数

(4) dirty_pages_misses
脏页缓存未命中的写操作数

(5) Read IOPS
每秒读操作数

(6) Write IOPS
每秒写操作数

(7) used_mb
使用的缓存空间

(8) unused_mb
空闲的缓存空间

(9) recliam_count
缓存回收次数

(10) cur_dirty_bytes
当前 OSC（对象存储客户端）上写入和缓存的字节数

(11) cur_grant_bytes
当前 OSC 与每个 OST（对象存储目标）保留的回写缓存空间

(12) inflight
待处理的 RPC 的数量

(13) timeouts
RPC 超时值

(14) avg_waittime
请求平均等待时间

(15) osc_cached_mb
当前总缓存空间

(16) req_waittime
请求在服务器处理之前在队列中等待的时间量

(17) req_active
现在正在处理的请求数

3.2 动作

动作决定了参数应如何调节，按离散动作空间来处理，为每个参数指定一个步长，则每个参数对应两个动作：调高和调低，调高即当前参数加步长，调低即当前参数减步长。此外，系统如果根据状态信息判断当前没有需要调节的参数，那么也应可以选择什么都不做。这样，动作空间即为：2×参数个数+1。

从安全的角度考虑，还应为每个参数设置范围，当调节后的值大于或小于该范围时，相应地设置值为最大值或最小值，以保证系统的正常运行。

当性能优化目标确定时，系统应针对该优化目标进行参数选取。这需要存储领域一定的先验知识，即需要知道所关注的性能变化是与哪些参数相关的。如果将存储系统的全部参数都纳入进来，动作空间会很大，模型不好收敛。本文的目标性能是吞吐率，选择的参数及其相关信息如下表：

表 1 调整参数信息

参数名	默认值	最小值	最大值	步长
max_dirty_mb（OSC 中可以写入多少 MB 脏数据）	32	32	4096	32
max_pages_per_rpc（在单个 RPC 中对 OST 进行 I / O 的最大页数）	256	64	1024	64
max_rpcs_in_flight（从 OSC 到其 OST 的最大并发 RPC 数）	8	4	256	4
max_read_ahead_mb（文件上预读的最大数据总量）	40	0	160	40
max_cached_mb（客户端缓存的最大非活动数据量）	128	128	32234	128
max_read_ahead_per_file_mb（每个文件预读的最大数据量）	40	0	80	10
max_read_ahead_whole_mb（可以完整读取的文件的最大大小）	2	0	8	2
statahead_max（statahead 线程预取的最大文件属性数）	32	0	8192	32

3.3 reward

reward 的设计是强化学习最关键之处，因为模型的训练是依赖 reward 进行的，reward 设计的好坏往往决定了一个强化学习算法最后能不能成功应用。

分布式存储系统的负载是不断变化的，如果只是简单定义 reward 为当前吞吐率与上一时刻吞吐率之差，当负载剧烈变化时，reward 会相应有很大变化，那么此时的 reward 系统无法分辨是因为负载变化导致，还是因为参数调节导致，导致模型无法收敛。

如果不只是考虑当前时间点的吞吐率跟上时间点的吞吐率差，而是考虑动作执行后某一时间段内的

吞吐率变化情况作为 reward，比如最近一定步数 N 或者一定时间窗口 W 内的吞吐率差，即：

$$\sum_{n=0}^{N}(\text{吞吐率差} \times \gamma^{n}) \quad (13)$$

这样当时间段选取合适的值时，可以克服某个时间点（或某个时间段）负载剧烈变化导致 reward 受影响的问题。γ 值的选取代表了是更关心当下的奖励还是长期奖励。

可能会出现这个时间窗口内，负载都一直在不断加大，从而导致 reward 因为负载的原因一直在变大，这样的情况可以接受的，并且 reward 也理应给高，因为这代表系统利用率高（除了吞吐率外也考虑系统的利用率）。

3.4 接口

接口模块介于强化学习算法模块与目标集群之间，负责两个模块之间的消息通信，使得两个模块可以独立开发而互相不影响，践行了强内聚、松耦合的设计模式。

本文的消息通信模块选用了 ZeroMQ，它是一种基于消息队列的多线程网络库，其对套接字类型、连接处理、帧、甚至路由的底层细节进行抽象，提供跨越多种传输协议的套接字，可并行运行，分散在分布式系统间。ZeroMQ 将消息通信分成 4 种模型，分别是一对一结对模型（Exclusive-Pair）、请求回应模型（Request-Reply）、发布订阅模型（Publish-Subscribe）、推拉模型（Push-Pull）。基于系统的架构，本文采用了请求回应模型。

3.5 模型

一个前馈神经网络如果具有线性输出层和至少一层具有任何一种“挤压”性质的激活函数（例如 logistic sigmoid 激活函数）的隐藏层，只要给予网络足够数量的隐藏单元，它可以以任意的精度来近似任何从一个有限维空间到另一个有限维空间的 Borel 可测函数，此即万能近似定理[4]。万能近似定理意味着无论我们试图学习什么函数，我们知道一个大的 MLP 一定能够表示这个函数。具有单层的前馈网络足以表示任何函数，但是网络层可能大得不可实现，并且可能无法正确地学习和泛化。在很多情况下，使用更深的模型能够减少表示期望函数所需的单元的数量，并且可以减少泛化误差。

深度学习与强化学习的结合，即用深度神经网络去逼近强化学习的值函数或策略函数。本文我们采用 pytorch 0.4 实现了含 3 个隐藏层的全连接神经网络，每个隐藏层的单元数量是状态空间的 2 倍，激活函数为 relu 函数，优化算法为 Adam 算法。

4 实验

4.1 实验环境

整个存储系统有 2 个服务器节点和 2 个客户端节点。2 个服务器节点中 1 个为元数据服务器（MDS）节点，1 个为对象存储服务器（OSS）节点，对象存储服务器管理着 22 个对象存储目标（OST）。这些服务器节点在测试期间均为闲置节点，以避免其他负载对测试结果的影响。2 个客户端节点采用相同的配置：8 个 Intel(R) Xeon(R) CPU E5620 @ 2.40GHz，32GB RAM，1TB SATA，网络带宽为 10000Mb/s。

策略节点配置有 24 个 Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz，64GB RAM，以及 2 个 NVIDIA Tesla K80 GPU。

本文的分布式文件系统选用了 Lustre（2.5.3）。强化学习算法使用了 Pytorch（0.4）实现。

4.2 测试工具

本文选用了 iozone（3.479）来进行测试。iozone 是一个文件系统的 benchmark 工具，可以测试不同的操作系统中文件系统的读写性能。

4.3 测试项

本文选取的各测试项及其定义如下：

(1) Write: 测试向一个新文件写入的性能。
(2) Re-write: 测试向一个已存在的文件写入的性能。
(3) Read: 测试读一个已存在的文件的性能。
(4) Re-Read: 测试读一个最近读过的文件的性能。
(5) Strided Read: 测试跳跃读一个文件的性能。
(6) Random Read: 测试读一个文件中的随机偏移量的性能。
(7) Random Write: 测试写一个文件中的随机偏移量的性能。

测试以吞吐量模式运行，指定每个客户端节点启动 16 个进程来并行读写。测试文件大小 8GB，文件块大小 1MB，跳读跨度为 2MB。测试时会在测试目录里生成各个节点的数据包，测试完成后在日志文件里会看到各个节点的读写速度，最大速度，最小速度，平均速度，还有总的吞吐量。

4.4 实验流程

(1) 采用 lustre 默认参数配置测试 3 次，取其测试结果均值作为 baseline。
(2) 启动 iozone 测试，独立地对每个算法训练一定的时间，将训练好的模型储存以备调用。
(3) 将系统参数重置为默认值。
(4) 启动 iozone 测试，同时启动调节系统，每个算法测试 3 次后，取其测试结果均值得到最

终结果。

4.5 实验结果

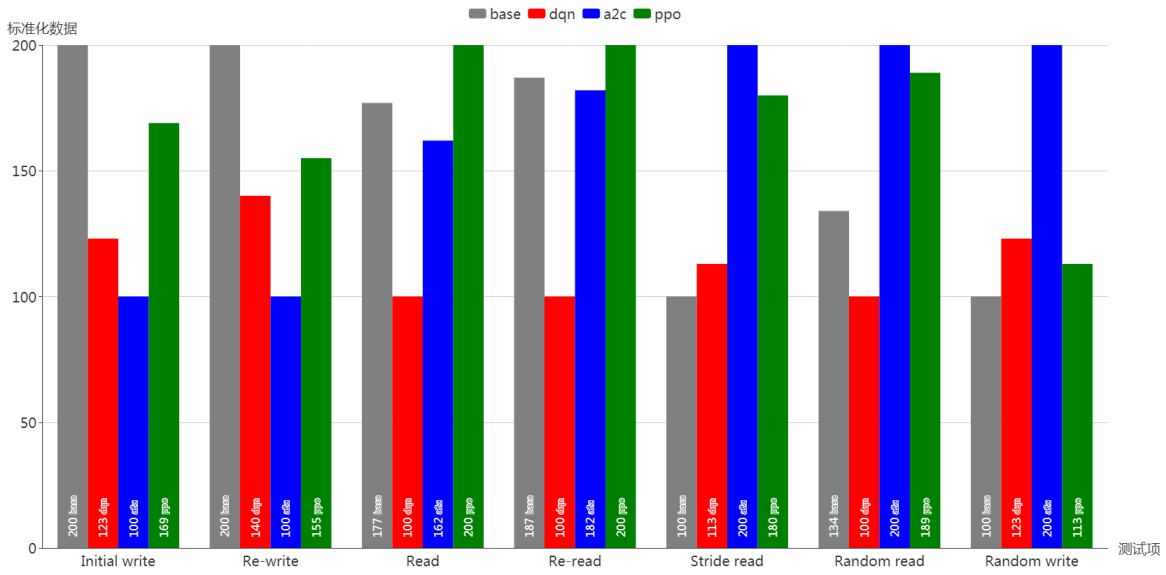

图 4 测试结果（测试数据标准化）

Fig.4 Test Results (Standardized Test Data)

搭建好测试环境后，我们对各强化学习算法进行了测试。测试前，对每个算法模型预先训练 24h，后对每个算法测 3 次（每次测试花费时长约 10h），取其均值作为测试结果，如表 2，测试数据为 32 个并发读写进程的平均吞吐率。为了更加直观地体现测试结果，我们采用直方图对测试数据进行可视化，但由于各测试项数据差异较大，致使数据值较小的测试项展示效果不太明显，故又对测试数据进行了标准化（范围 100-200），如图 4 所示。

可以看到，在跳读和随机读测试项上，a2c 和 ppo 强化学习算法都有明显的提升效果。而 dqn 算法在这种负载多变的测试环境上表现较差，这在李的论文中也有所体现[7]，其测试结果指出 dqn 算法在测试读写比例为 1:1 时，性能几乎没有提升，本文的测试读写比例即为 1:1。因此，针对存储系统参数调优任务来说，策略梯度方法是明显优于值函数方法的。具体来说，虽然 ppo 算法对 5 个测试项都有性能提升，而就高能物理计算环境所关注的跳读和随机读来说，a2c 算法表现更好，是最贴近实际应用的算法。

经过对测试结果的分析，我们发现使用强化学习的方法来对存储系统的参数进行调节，确实对存储系统性能有了较为显著的提升，而当负载发生变化时，其可进行动态适应性的调整，并且对生产系统的影响很小。可以预见，将其部署在生产系统后，在提升系统性能的同时，可大大减少人力成本及时间成本。

表 2 测试数据（进程平均吞吐率，单位：KB/S）

算法 / 测试项	Base	DQN	A2C	PPO
Initial write	1233	1186 (-4%)	1172 (-5%)	1215 (-1%)
Re-write	1249	1202 (-4%)	1171 (-6%)	1214 (-3%)
Read	38613	30505 (-21%)	36980 (-4%)	40924 (+6%)
Re-read	41777	30825 (-26%)	41241 (-1%)	43380 (+4%)
Stride read	3493	3570 (+2%)	4051 (+16%)	3941 (+13%)
Random read	1893	1566 (-17%)	2505 (+32%)	2407 (+27%)
Random write	560	565 (+1%)	581 (+4%)	563 (+1%)

5 总结与展望

本文介绍了一种基于强化学习的参数调节方法，实验表明该方法使我们所关注的性能得到了显著提升。事实上该方法不只是针对分布式存储领域，通过自定义环境和奖励，它可以泛化到更多的领域中。

总的来说，我们将强化学习应用到分布式存储领域中只是一次初步的尝试，有很多方面需要进一步探

索：

(1) 本文只是对 lustre 客户端的参数进行了调节，未来我们会增加 lustre 服务器端以及其他分布式文件系统（如 eos，ceph）[1]的参数调节。

(2) 状态的表示。本文我们将当前时间点跟系统性能相关的因素作为状态来训练，事实这可能存在一定的局限，只了解到当前的状态，而对系统的历史信息以及变化趋势没有涉及。因此，未来的工作会对系统的状态表示加以改进。

(3) 算法。强化学习发展迅速，不断有新的方法被提出，如逆向强化学习，分层强化学习，世界模型等，新的算法的性能往往会有较大提升，我们会在未来的工作中加深对算法的研究以及应用，并在前人的基础上尝试提出自己的强化学习算法。

参 考 文 献

[1] CERN. *http://eos-docs.web.cern.ch/eos-docs/*. On-line Resources, 2017.

[2] Haifeng Chen, Guofei Jiang, Hui Zhang, and Kenji Yoshihira. Boosting the performance of computing systems through adaptive configuration tuning. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1045–1049. ACM, 2009.

[3] Yixin Diao, Joseph L Hellerstein, Sujay Parekh, and Joseph P Bigus. Managing web server performance with autotune agents. *IBM Systems Journal*, 42(1):136–149, 2003.

[4] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[5] Jaehyun Han, Deoksang Kim, and Hyeonsang Eom. Improving the performance of lustre file system in hpc environments. In *Foundations and Applications of Self* Systems, IEEE International Workshops on*, pages 84–89. IEEE, 2016.

[6] Pooyan Jamshidi and Giuliano Casale. An uncertainty-aware approach to optimal configuration of stream processing systems. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016 IEEE 24th International Symposium on*, pages 39–48. IEEE, 2016.

[7] Yan Li, Oceane Bel, Kenneth Chang, Ethan L. Miller, and Darrell D. E. Long. Capes: Unsupervised storage performance tuning using neural network-based deep reinforcement learning. In *Supercomputing '17*, November 2017.

[8] lustre. *http://lustre.org/documentation/*. On-line Resources, 2017.

[9] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[11] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[13] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[14] Fan Zhang, Junwei Cao, Lianchen Liu, and Cheng Wu. Performance improvement of distributed systems by autotuning of the configuration parameters. *Tsinghua Science and Technology*, 16(4):440–448, 2011.

Zhang Wentao, born in 1993. Master. His research interests include reinforcement learning, machine learning and mass storage.

Wang Lu, born in 1983. PhD, Associate Research Fellow. Her main research interest include mass storage, machine learning and deep leaning.

Cheng Yaodong, born in 1977. PhD, Associate Research Fellow. His main research interest include cloud computing, mass storage and grid computing.