第47卷 第7期 计 算 机 工 程 2021年7月
Vol.47 No.7 Computer Engineering July 2021

·体系结构与软件技术· 文章编号：1000-3428(2021)07-0205-07 文献标志码：A 中图分类号：TP391

基于DPDK的高速存储I/O优化方法

朱文俊[1]，徐 壮[1]，秦家佳[1]，李 鹏[1,2]

（1.南京邮电大学 计算机学院，南京 210023；2.江苏省高性能计算与智能处理工程研究中心，南京 210023）

摘 要： 网络I/O在Redis存储过程中是限制存储性能的关键因素，而默认参数或人工参数配置会制约存储性能。针对参数配置不当导致存储吞吐性能下降及时延较高的问题，提出一种存储I/O优化方法GTS。考虑各阶段参数对存储性能的影响，在DPDK的优化原理基础上通过分析处理特性，采用分层模型策略实现对存储性能预测，从而寻找出最优参数调优方案。实验结果表明，与默认参数相比，GTS方法能够有效提升存储吞吐量，且在写密集下较ATH算法具有更低的时延。

关键词： DPDK技术；性能调优；内存数据库；参数配置；ATH算法

开放科学（资源服务）标志码（OSID）：

中文引用格式： 朱文俊，徐壮，秦家佳，等.基于DPDK的高速存储I/O优化方法[J].计算机工程，2021，47(7)：205-211，217.

英文引用格式： ZHU W J，XU Z，QIN J J，et al.DPDK-based optimization approach for high speed storage I/O[J].Computer Engineering，2021，47(7)：205-211，217.

DPDK-Based Optimization Approach for High Speed Storage I/O

ZHU Wenjun[1]，XU Zhuang[1]，QIN Jiajia[1]，LI Peng[1,2]

（1.School of Computer Science，Nanjing University of Posts and Telecommunications，Nanjing 210023，China；
2.Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks，Nanjing 210023，China）

【Abstract】 In the Redis storage process，I/O is a key factor that influences storage performance. Inaccurate default parameter configuration or manual parameter configuration will restrict the performance of storage and increase the delay. To address the problem，an optimization method for storage I/O is proposed. The method fully considers the impact of parameters of each stage on storage performance.Based on the optimization principle of DPDK，the processing characteristics are analyzed，and the layered model is established to effectively predict the storage performance to find the optimal parameter configuration. Experimental results show that the GTS algorithm can effectively improve the storage throughput compared with the default parameters，and reduce delay compared with the ATH approach in write-intensive conditions.

【Key words】 DPDK technology；performance tuning；memory database；parameter configuration；ATH algorithm

DOI： 10. 19678/j. issn. 1000-3428. 0058139

0 概述

内存数据库因具有低延时的数据访问特点，已成为当前互联网服务重要的加速引擎。然而，DPDK[1]技术的出现以及批次处理[2]技术的广泛应用，给内存数据存储带来了新的要求。目前主流的内存数据库有Amazon提出的Dynamo[3]、Facebook提出的Memcached[4]、Facebook与Yahoo！共同完善的HBase[5]。在上述系统中，虽然为数据存储提供了极高的读/写操作性能和高效的数据管理服务，但是面对高吞吐、高并发、高负载的复杂网路存储环境，如何配置运行参数来调优内存数据库的性能，成为网络数据存储的一项重要挑战。由于DPDK与Redis[6]在协同存储过程中依赖于程序人员的经验配置来满足不同场景下的网络存储要求，因此会出现资源参数配置不足和资源参数配置过剩的问题。虽然Redis提供了几种不同的数据分发策略，例如数据分片[7]、Hash slots[8]等方式来优化存储的性能，但是

基金项目： 国家重点研发计划（2018YFB1003201）；国家自然科学基金（61672296，61602261，61872196，61872194，61902196）；江苏省科技支撑计划项目（BE2017166，BE2019740）；江苏省高等学校自然科学研究重大专项（18KJA520008）；江苏省“六大人才高峰”高层次人才项目（RJFW-111）。

作者简介： 朱文俊（1992—），男，硕士研究生，主研方向为网络安全；徐 壮、秦家佳，硕士研究生；李 鹏，教授、博士。

收稿日期： 2020-04-22 **修回日期：** 2020-06-28 **E-mail：** jue99@qq.com

数据分发的策略受到系统参数的直接影响,而采用默认参数配置无法适应多变的应用环境,尤其在DPDK的高速网络数据存储情况下更为明显。在高达10 Mpacket/s[9]以上的速率时,如何配置合理的存储参数,从而满足系统需求是解决该问题的重要方向。在存储过程中吞吐量和时延是表现数据存储效率的重要特征,而采用默认参数配置无法正常发挥数据库的存储性能,使得存储吞吐与时延无法符合实际应用场景的需求。

而解决此类性能优化问题的最主要途径是通过对Redis系统参数的分析,来寻找到最优的资源配置,从而使得DPDK下不同网络数据存储都能达到最优的性能。

本文通过形式化分析DPDK下的Redis存储I/O优化模型,根据批次大小等存储参数对性能的影响层度,分层建立性能感知模型。在此基础上,通过遗传算法[10]迭代寻找出最优的参数方案,从而根据环境变化调整参数配置,提升存储I/O吞吐量,降低操作过程的时延,达到优化整体性能的目的。

1 相关工作

随着DPDK技术的出现,网络数据呈现多样化发展。与传统技术相比,所需存储的数据规模、数据复杂度、数据完整性对网络数据存储系统的读写性、可用性、有效性都提出极高的性能需求。而参数自动调优技术是高速存储I/O优化的重要研究方向,参数调优技术是指在一系列参数特征中,通过某种性能度量或者反馈机制,寻找出当前环境下最优的配置项组合。

目前常用的存储统参数调优方法包括3个方面:控制反馈方法,参数搜索方法和统计学习[11]方法。控制反馈方法[12]是基于参数配置规则的动态反馈来控制参数配置,主要方式是通过性能监控套件[13-14]或者建立相应规则来根据运行情况做出调整,如文献[15]提出一种名为PCM的基于策略驱动的存储系统配置系统,该系统可以捕获不同工作负载的调整参数作为配置策略。在多种工作负载下,PCM配置下的工作性能优于默认配置,使得吞吐量显著提高。但是这种基于策略驱动的优化方法在运行中需要频繁进行策略更替,提高了时间成本。参数搜索法[16]则是将参数配置作为黑盒处理,使用搜索算法针对特定系统的特定性能情况进行搜索,当搜索环境发生变化时该方法不适用,并且在优化参数配置时每次都需要运行具有大量输入数据集的应用程序,会耗费大量时间和系统资源。另一种方法是基于统计学习的优化方法,其通过已有的配置项和性能组成的数据集样本进行学习并训练性能模型。该方法能够有效实现对性能预测,但是性能模型建立的准确性会直接影响到优化效果,并且特征过多导致样本数据过大也是该方法的弊端。文献[17]将系统参数配置与机器学习进行有效结合,提出一种根据不同的学习体系结构来促进配置调整任务的优化方法,其中参与者和批评者都由多层神经网络实现,并利用误差反向传播算法[18]根据学习中产生的时间差误差调整网络权重,但是该方法局限于只能针对单一服务器进行,在面对多个节点服务器情况时会失效。文献[19]提出了一种给定的工作负载下应用程序自动调整配置参数的新方法,称为ATH。该方法构建以配置参数为输入的低成本精确性能模型,有效提升了系统的吞吐量,但是该方法中的模型以时延和吞吐通过线性加权的方式进行构建,缺少对存储流程以及历史作业的分析,缺少参数与参数之间关联性的考虑。文献[20]提出一种AutoConfig算法,使用抽样的方法将大量配置项抽样出多个参数,提高了建模的性能。短板在于没有考虑到多个配置项之间的关系,对于2个具有相关性的参数,其通常会考虑选择其中1个。

上述方法能在一定程度上解决存储系统繁琐的参数配置调整问题,但未能考虑存储调优中模型建立的非线性关系以及参数特征数目巨大问题,因此,需要一种更加精确的性能优化算法,使得在DPDK下以Redis为代表的内存存储系统利用能够充分考虑参数之间的相互作用关系,提升调优效率。本文提出一种存储I/O调优方法,通过特征筛选以及建立基于作业特性的渐进感知模型,解决默认参数导致的性能瓶颈问题。

2 存储参数优化算法设计

2.1 问题建模

解决DPDK下内存存储参数配置问题目的是寻找出最优的配置方案,使得存储的吞吐量与时延达到当下应用环境的近似最优值。对于Redis本身架构而言,整体性能的衡量是通过应用端和服务端2个方面来进行评估。在要素特征上分析主要包括写入比例、请求交方式、存储数据大小、请求并发量等因素。

假设Redis集群中主节点有M个,则节点的集合为$M=(m_1,m_2,\cdots,m_M)$,且每个节点中资源受到集群的内存、网络、存储等方面影响,即$S_{to}=(E_j,O_j,S_j)$,E_j表示该集群中第j个工作节点提供的内存资源,O_j为节点j上可以提供的网络资源,S_j为节点j提供的存储资源。此外,需要评估数据库中的负载情况,则具体通过W_j表示第j个节点的工作负载。其中,由于DPDK利用批次和管道处理来优化数据存储,因此设定管道一次请求数为B^S而总数据接收批次为B^T,则整个存储过程中数据的有效批次比例如下:

$$B_p^E=\frac{B^S}{B^T} \tag{1}$$

在数据库的工作负载上,设定N_j^r表示读请求数据大小,N_j^W表示写请求数据大小,在第j个节点上的

读请求比例为P_j^r，写请求比例表示为P_j^w，那么读写比例为P_j^r/P_j^W。

本文所考虑的场景是高速网络环境的数据存储，所以针对的应用场景为写密集。因此，设定请求数据为$P_j^w \approx 100\%$，并且假设Redis数据库中各个节点在数据写入中的分配比为A_j，抽象出的资源需求公式化表达为：

$$\sum_{j=1}^{m}\left[W_j \times N_j^W \times P_j^W \times A_j\right] \times B_p^E \leqslant \sum_{j=1}^{m}\left[E_j, O_j, S_j\right] \quad (2)$$

从式(2)可以看出，基于DPDK的高速网络存储机制是依赖于多种因素的影响，且整个资源需求受限于总体资源的限制，而总体资源又受到Redis与DPDK的配置参数直接影响。

那么对存储I/O性能优化问题，即性能最优值与参数配置关系可以定义为：

$$\max imize \quad S_{SPF}(X) \quad (3)$$

其中：X表示DPDK与Redis的配置参数组合；SPF则是需要获取的性能模型，代表了具体应用下配置参数与应用性能之间的关系。

本文针对DPDK下Redis存储的性能优化问题，设计一种渐进感知模型，其核心在于根据各阶段的历史数据来分层构建性能模型。将底层参数与应用层参数的相互作用考虑到模型构建中，从而改进传统的加权回归预测模型，并且将所得模型与遗传算法相结合形成GTS配置优化算法。遗传算法可以在计算预测模型时有效避免局部最优解的产生，相较于粒子群算法以及经典的最小二乘法(Ordinary Least Squares，OLS)，更加适合渐进感知模型的特性。

2.2　算法设计

本文提出的GTS算法主要分为3个部分：第1部分是进行特征筛选，该部分在获取原始特征基础上使用ANOVA进行正交实验，生成一定配置组合参数，运行在Redis与DPDK上使其生效，并且通过测试框架模拟高速负载的存储环境，从而获取初始的性能数据集；第2部分是进行性能模型的构建与训练，在这部分中改进了常用的加权回归模型，将底层DPDK参数和应用级Redis参数与性能建立渐进感知模型，通过多个局部模型的训练进一步生成系统整体模型，充分考虑参数与参数的关联性；第3部是性能优化，将第2部分所得渐进感知模型与遗传算法相结合，通过多次迭代寻找出性能最优解以及最优参数配置。整体优化架构如图1所示。

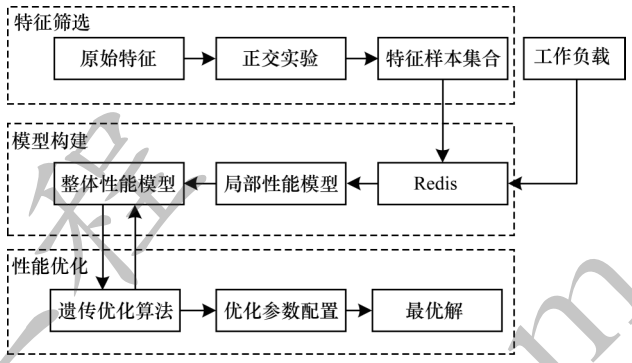

图1　整体优化框架

Fig.1　Overall optimization framework

3　存储参数优化方法

本节对本文所提出的存储I/O调优方法的具体实现进行叙述，该方法通过使用ANOVA对存储中涉及的特征进行筛选，并且改进原有线性回归的建模思想，考虑模型中各部分参数的相互影响。存储参数调优算法主要分为特征筛选、渐进感知模型构建及GTS优化算法。

3.1　特征筛选

对DPDK下内存数据的存储配置主要分为2个部分：DPDK配置和Redis配置。而对于两者的特征筛选，主要是对Redis进行敏感性识别，原因在于底层程序配置参数范围更加容易确定。Redis的配置参数主要包含General、Snapshotting、Limits、Lua Scripting、Replication等[21]。而系统的参数对于系统的性能有着不同层度影响，Redis系统部分配置描述如表1所示。

表1　Redis系统部分参数配置

Table 1　Partial parameters configuration of Redis system

效用范围	参数	描述
General	tcp-backlog	确定已完成队列的长度，通过调整可以提升客户端反应速率
	databases	设置数据库的数量
Limits	maxclients	设置同一时间内最多连接数
	maxmemory <bytes>	设置最大内存
Redis Cluster	cluster-node-timeout	设置集群节点超时时间
Lua Scripting	lua-time-limit	设置Lua脚本的最大运行时间，单位为ms
Advanced Config	hash-max-ziplist-value	设置哈希压缩列表最大阈值
	hz	配置可以优化延时
	client-output-buffer-limit	客户端缓冲区最大占用阈值
	hll-sparse-max-bytes	存储数据结构选择界定值

为能够有效地获取Redis中各项元素对数据存储能力的敏感性，利用ANOVA分析法，针对每一个元素，通过多次实验来获取性能的感知矩阵：

$$\boldsymbol{a}_{m\times n}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \tag{4}$$

通过矩阵可以得出，每行元素的平均值和性能总平均值如式(5)和式(6)所示：

$$T_i=\frac{\sum_{j=1}^{n}a_{ij}}{m} \tag{5}$$

$$\overline{a}=\frac{\sum_{i=1}^{m}T_i}{m} \tag{6}$$

根据每行的平均值和性能总平均值，可以通过式(7)、式(8)计算出效应平方和S_A以及误差平方和S_E：

$$S_A=\sum_i\sum_j\left(T_i-\overline{a}\right)^2 \tag{7}$$

$$S_E=\sum_i\sum_j\left(a_{ij}-T_i\right)^2 \tag{8}$$

根据所得效应平方和与误差平方和可以通过式(9)计算出总误差变差为S_T：

$$S_T=S_A+S_E \tag{9}$$

其中：自由度为$f_T=N-1$，$f_A=m-1$，$f_E=f_T-f_A$，由此可以进一步计算出$\overline{S_A}=\frac{S_A}{f_A}$和$\overline{S_E}=\frac{S_E}{f_E}$，并且可以得出显著水平$\alpha$。通过比较判断各元素的灵敏度，从而生成参数配置集合，并将生成的参数集合运行在相应的存储环境，收集相应的性能数据。

3.2 模型构建

在高速网络存储配置过程中影响网络I/O的主要因素包括DPDK驱动配置和存储应用配置2个方面。整体的配置活动图如图2所示。

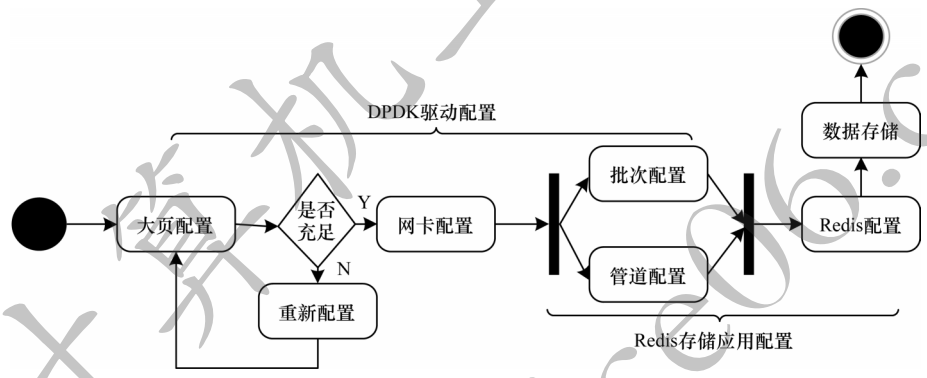

图2 DPDK与Redis参数配置活动图

Fig.2 Activity diagram with parameter configuration of DPDK and Redis

因此，整体程序的性能可以2个变量进行表示，式(1)可以进一步表示为：

$$S_{\mathrm{SPF}}(X)=S_{\mathrm{SPF}}(C_{\mathrm{DPDK}},C_{\mathrm{Redis}}) \tag{10}$$

其中：S_{SPF}作为性能指标的计算；C_{DPDK}表示图2中网卡以及批次、管道等配置集合[22]；C_{Redis}则由运行参数配置、集群参数、限制参数等多个集合的组合。

为能够实现对系统性能有效感知和评估，需要建立准确的性能模型。而现有的模型构建方法单纯地采用多个参数求和的方法，基于DPDK的高速流量存储机制并不完全处在同一个水平面上，采用传统模型构建方法忽略了参数之间的影响关系。底层DPDK驱动参数的各项配置应该与Redis性能之间产生关系。因此，本节采用一种渐进模型来实现对系统性能预测，该模型的核心在于采用分层建立的方法，建立局部回归模型，并根据局部回归模型得到局部预测值，局部模型测量值作为整体系统回归模型的参数与整体参数重新训练，从而逐步得到整体的性能模型。为了降低系统模型训练的成本，对于局部模型的建立基于阶段的执行实现，并建立时间模型。模型可以表示为：

$$T=f(C),C\in(c_1,c_2,\cdots,c_n) \tag{11}$$

其中：$f(C)$是基于梯度提升树的性能模型。该模型将各阶段的配置参数作为输入，性能指标延迟作为输出。在此说明，式(11)所示为针对局部建立的数字模型，并不包含一个具体准确的表达公式。T的建立通过GBDT算法训练得到。因此，样本可以表示为：

$$G_{\mathrm{GBDT}}(T_i,C_i),i\in(i_1,i_2,\cdots,i_n) \tag{12}$$

其中：T_i是第i个样本中执行时间的值；C_i是第i个样本中各工作阶段的参数对应的值。在获取到局部模型后，通过迭代算法进行训练，根据模型所感知的值重新作为参数进一步训练出整体性能模型。因此，整体模型的定义如下：

$$\left[\begin{array}{l} \underbrace{\left\{\mathrm{CSV}_{\mathrm{DPDK}}\right\}\xrightarrow{\mathrm{train}}\left\{T_{\mathrm{DPDK}}\right\}}_{\text{local phase}} \\ \\ \underbrace{\left\{\mathrm{CSV}_{\mathrm{Redis}}\right\}\xrightarrow{\mathrm{train}}\left\{T_{\mathrm{Redis}}\right\}}_{\text{local phase}} \end{array}\right]\rightarrow$$

$$\left\{\mathrm{CSV}_{\mathrm{DPDK}},\mathrm{CSV}_{\mathrm{Redis}},T_{\mathrm{DPDK}},T_{\mathrm{Redis}}\right\}\xrightarrow{\mathrm{train}}\left\{T_{\mathrm{system}}\right\} \tag{13}$$

其中：$\mathrm{CSV}_{\mathrm{DPDK}}$表示DPDK运行过程的性能数据文

件；T_{DPDK} 表示DPDK阶段的性能模型；$\mathrm{CSV}_{\mathrm{Redis}}$ 表示Redis下运行过程的性能数据文件；T_{Redis} 为存储阶段的性能模型；T_{system} 表示系统的最终的渐进感知模型。渐进感知模型的构建流程如图3所示。

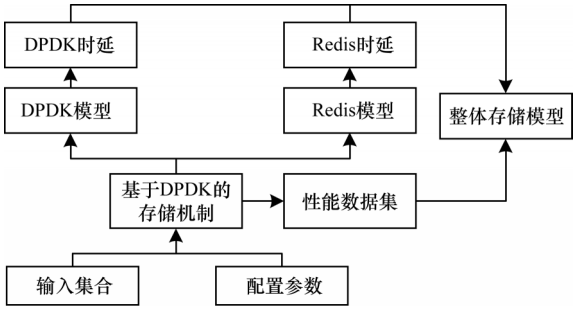

图3　渐进感知模型的构建流程

Fig.3　Construction process of progressive perception model

从渐进模型的构造流程可以看出，通过构建DPDK模型与Redis模型可以评估出 T_{DPDK}、T_{Redis}，并将 T_{Redis}、T_{DPDK} 与系统所有参数进行训练得到系统的整体时延模型。DPDK模型主要是对数据接收到数据分发过程的执行时间模型；Redis模型则是数据库连接以及数据存储整个过程的执行时间模型。

在存储机制下存储资源的优化过程中，DPDK驱动参数配置与Redis存储应用参数配置之间有着无法直接推断的关系，在缺乏整体的分析和详细组合的情况下，很难直接给出精准的数据公式，因此需要通过算法进行训练，本文采用GBDT算法对所涉及的模型进行训练。在GBDT算法中，需要输入负载应用程序[23]的产生数据集合 D、配置参数集合 $C_{\mathrm{system}}=\{C_{\mathrm{DPDK}},C_{\mathrm{Redis}}\}$ 以及有底层启动到应用数据库存储的各部分时间集合 T_P。对于样本 N，从 $j=1$ 到 N 计算出残差如式(14)所示：

$$r_{mi}=-\left[\frac{\partial L(y_i,f_i(x))}{\partial f_i(x)}\right]_{f_i(x)=f_i^{j-1}(x)} \tag{14}$$

对所得残差 r_{mi} 进行拟合得到拟合残差 $t_i^j(x)$，并根据拟合残差值，通过式(15)更新模型：

$$f_i^j(x)=f_i^{j-1}(x)+t_i^j(x) \tag{15}$$

通过该方法可以训练出DPDK与Redis阶段的局部模型 T_{DPDK}、T_{Redis}，并将所得预测值作为整体模型的输入参数进行重新训练。需要注意是，在对整体模型的训练中，输入集合需要增加存储总时间PF，因为此次训练过程中针对程序运行的整体训练，与局部模型相比需要将拟合出的决策树重新加入到模型中去，从而根据式(16)得到整体模型：

$$T_{\mathrm{system}}(d,c)=\sum_{j=1}^{N}t_{\mathrm{system}}^j(d,c,t_{\mathrm{DPDK}},t_{\mathrm{Redis}}) \tag{16}$$

上述是模型构建的具体过程，可以看出整体训练过程分为2个阶段：第1阶段是针对局部的训练；第2阶段主要是结合局部重新对残差计算并且拟合得到拟合残差 $t_{\mathrm{system}}^j(x)$ 从而获得整体的模型。因此，本文提出的渐进性能感知模型是一种考虑到系统局部与整体关系的分层模型，并且通过GBDT算法能够在模型参数调整较少时，训练出精准度更高的模型，所以利用梯度提升回归树算法来针对渐进感知模型获取资源配置调优中的重要参数。

3.3　GTS参数配置优化算法

在建立有效的渐进感知模型后，需要基于渐进感知模型对系统性能进行优化。遗传算法是一种用于求解此类问题的搜索算法，该算法能够有效地避免陷入局部最优解的问题。因此，本文将提出的渐进感知模型与遗传算法有效结合，形成GTS(Gradual Tuning Storage)配置优化方法，提高问题求解质量，寻找出最优的资源分配方案。优化算法迭代最优配置的步骤如图4所示。

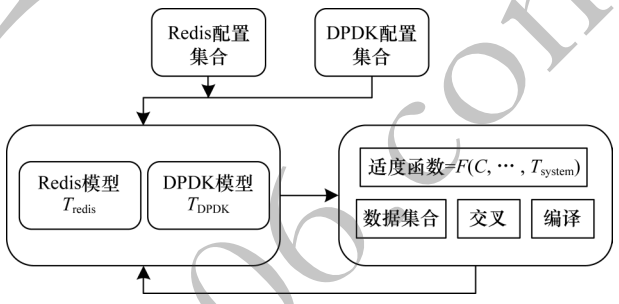

图4　迭代最优配置过程

Fig.4　Process of iterating optimal configuration

从图4可以看出，将得到局部模型生成的渐进感知模型和参数集合作为输入，进行资源配置优化，通过多次的迭代训练搜索出最优的资源配置方案。详细实现如算法1所示。

算法1　GTS算法

输入　配置集合 C，渐进感知模型 T_{system}，适度函数 F，交叉概率 P_0，变异率 P_{m0}，停止规则 R

输出　最优组合 C_{new}

1.C′←ANOVA— C进行配置集合约束

2.初始化种群规模popsize，交叉概率 P_0，变异概率 P_{m0}；

3.H = random(popsize)随机生成 k 个组配置集合作为算法个体

4.While R！ do

5.For i < k do

6.fitness←— F(h, T_{system})计算每一组的适应值；

7.Sort(H)按适应度对种群个体进行排序；

8.按照适应度选择个体，放入新的种群；

9.h_i ←— p_O 交叉生成后代；

10.h_i ←— p_{mo} 个体变异；

11.将生成的个体加入到新的种群；

12.End for

13.C' = $C_1 \times C_2, \cdots, C_k$ 选取最优组合

14.End while

15.Return最优组合 C_{new}

4 实验与结果分析

4.1 实验环境

为验证DPDK下面向Redis的存储资源配置优化的有效性，本文实验的环境构建在集群之上，使用的Redis版本为Redis-4.0.9，是目前较为稳定的版本。测试集群建立在2台服务器上，服务器的相关信息如表2所示。

表2 测试机器的参数配置

Table 2 Parameter configuration of test machine

参数	测试机1	测试机2
IP	10.166.38.3	10.166.38.4
OS	Ubuntu 16.04.1	Ubuntu 16.04.1
CPU	Intel® Xeon® CPU E5-2620 v4	Intel® Xeon® CPU E5-2620 v4
主频/GHz	2.10	2.10
内核数	8	8
线程数	16	16
内存/GB	384	384
网卡/GbE	10	10

因为实际物理服务器的个数难以到达实验所需规模的集群，及采用1机1实例，所以需要在已有的2台物理服务器中开启多个Redis服务节点来模拟出大规模的Redis集群，对于初步节点的设置数，由于受到CPU的限制且一般Redis集群的建立采用奇数，因此2台服务器建立的集群规模为$2\times(16-2)-1=27$个节点的集群。

测试使用关键软件除了Redis 4.0.9外，为了测试高速网络下的存储以及集群存储下的整体配置优化，还使用测试工具MoonGen来生成测试数据集，所使用版本是基于DPDK-19.05的开发版本，软件搭建在单独一台物理服务器之上，防止程序之间的互相干扰。

使用YCSB（Yahoo! Cloud Serving Benchmark）模拟不同负载测试参数优化后的存储性能。YCSB是一个用来测试在线数据库性能与扩展性的框架。用户自主编写代码来测试数据库，也可以通过配置文件来指定需要进行负载类型测试，例如读写比例、记录大小、并发数等。

4.2 实验结果

为验证配置优化方法对系统整体性能调优的效果。实验选取存储吞吐量以及时延作为评估优化效果的标准。在实验过程中根据YCSB中每个基准测试程序，初始化DPDK程序以及Redis内存数据库，停止Redis集群，并且更新Redis运行配置和集群配置，重启Redis集群存储并启动Redis的基准测试，重复此过程并保证所有配置组合均被测试。

为对比各方法对系统存储时延的影响，采用不同配置优化算法进行重复实验。在实验过程中对于相同的算法，调整工作执行量进行多次实验。通过读取写密集下不同工作量的时间进行实验验证本文提出算法的有效性。在实验过程中对数据库采用Redis本身的Redis-cli辅助Redis-live进行监控，将数据发送的时间作为初始时间，并将读取数据显示时间作为完成时间，通过计算两者之间的差值，得到时间效率的实验结果。

针对不同负载环境下默认配置导致吞吐性能下降的问题，采用YCSB进行不同的负载类型的模拟实现，包括RMW、UH、RO、RH、AVG，其分别表示读写混合型、写密集型、只读型、读密集型、平均。图5反映了不同操作数下存储吞吐差异。

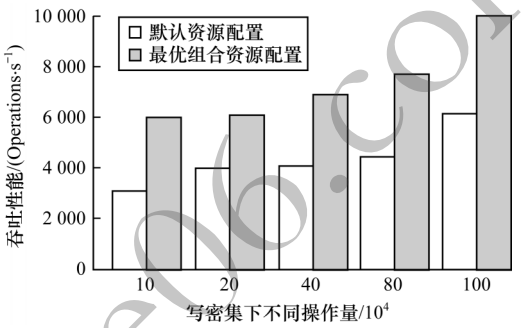

图5 写密集下I/O吞吐的性能对比

Fig.5 Performance comparison of I/O throughput under write-intensive

在图5中分别选取10万、20万、40万、80万、100万这5个层度进行测试，可以看出算法所得最优组合配置稳定适应操作数的变化并且不断上升在100万时可以到10 000 Operations/s，而采用默认配置运行差距较大。

图6所示主要是时延与不同操作数的关系，通过对比默认配置、ATH算法和本文的GTS算法可以看出，默认配置在不同的操作数10~80万下相对其他算法都表现出了较高的平均时延，本文的GTS算法与ATH算法相比在整体时延上有所降低。除写密集下不同负载的时延与吞吐测试以外，针对模型构建以及最优配置调整对速率影响进行实验，并对MoonGen产生的流量和实际处理速率进行对比。通过图7对比可以看出，由于两者监测有一定的延迟，因此整体波形向右偏移，虽然相对于MoonGen而言有着一定的成本消耗，但是整体的速率基本于MoonGen的测试速率保持一致，体现了本文优化方法具有较低的成本消耗。

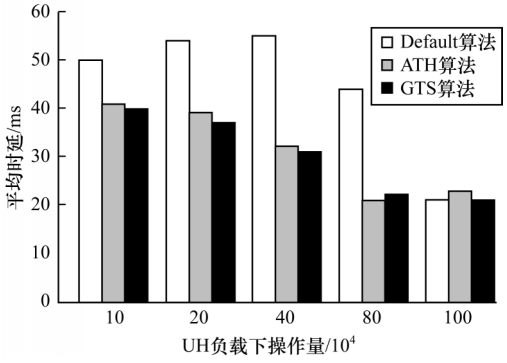

图6　写密集下不同算法的时延对比

Fig.6　Comparison of different algorithms in latency under write-intensive

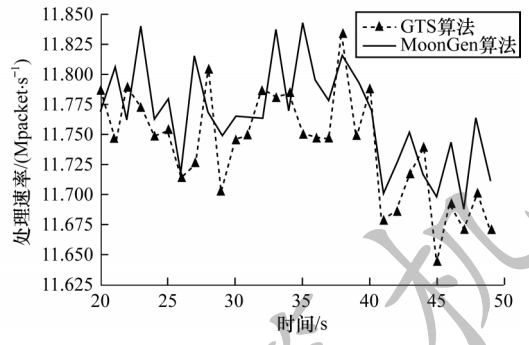

图7　测试速率与实际存储对比

Fig.7　Comparison of test rate and actual storage

5　结束语

本文针对默认资源配置参数限制高速网络存储效率的问题，建立性能模型，综合考虑资源配置对系统的影响，将渐进感知模型与遗传算法相结合，形成DPDK下面向Redis的资源配置优化方法，将局部模型作为基础生成系统模型，考虑内核级配置和应用级配置区别，提高模型的准确性，形成系统的最优配置方案。实验结果表明，该方法能有效优化资源配置参数，从而适应高速网络下的集群存储。下一步考虑将抽样技术与参数配置调优技术相结合，以提高存储I/O的效率。

参考文献

[1] 唐宏，柴卓原，任平. DPDK应用基础[M]. 北京：人民邮电出版社，2016.
TANG H，CHAI Z Y，REN P. DPDK application basis[M]. Beijing：Posts & Telecom Press，2016.（in Chinese）

[2] MIAO M，CHENG W X，REN F Y，et al. Smart batching：a load-sensitive self-tuning packet I/O using dynamic batch sizing[C]//Proceedings of the 18th IEEE International Conference on High Performance Computing and Communications. Washington D. C.，USA：IEEE Press，2016：726-733.

[3] DECANDIA G，HASTORUN D，JAMPANI M. Dynamo：Amazon's highly available key-value store[J]. ACM SIGOPS Operating Systems Review，2007，41(6)：205-220.

[4] CHENG W X，REN F Y，JIANG W C，et al. Modeling and analyzing latency in the memcached system[C]// Proceedings of the 37th IEEE International Conference on Distributed Computing Systems. Washington D. C.，USA：IEEE Press，2017：346-358.

[5] WANG K，LIU G L，ZHAI M，et al. Building an efficient storage model of spatial-temporal information based on HBase[J]. Spatial Science，2018，64(5)：1-17.

[6] LEE T，KIM Y，HWANG E. Abnormal payment transaction detection scheme based on scalable architecture and Redis cluster[C]//Proceedings of International Conference on Platform Technology and Service. Jeju，South Korea：[s. n.]，2018：1-6.

[7] 李翀，刘利娜，刘学敏，等. 一种高效的Redis Cluster的分布式缓存系统[J]. 计算机系统应用，2018，27(10)：91-98.
LI C，LIU L N，LIU X M，et al. High efficient distributed cache system based on Redis cluster[J]. Computer Systems &Applications，2018，27(10)：91-98.（in Chinese）

[8] CHEN S S，TANG X X，WANG H W，et al. Towards scalable and reliable in-memory storage system：a case study with Redis[C]//Proceedings of 2016 IEEE Trustcom/BigDataSE/ISPA. Washington D. C.，USA：IEEE Press，2016：1660-1667.

[9] DANIEL T，PETER S D，ROBERT O. PKTGEN：measuring performance on high speed networks[J]. Computer Communications，2016，82：39-48.

[10] AHMAD H，PRASATH V，MOHAMMED A，et al. An improved genetic algorithm with a new initialization mechanism based on regression techniques[J]. Information，2018，9(7)：167-196.

[11] DING Y，ZHAO Y，MA X B. A novel response-surface-based method in storage performance prediction[C]// Proceedings of Annual Reliability and Maintainability Symposium. Orlando，USA：[s. n.]，2019：1-5.

[12] DIAO Y X，HELLERSTEIN J L，PAREKH S，et al. Managing Web server performance with AutoTune agents[J]. IBM Systems Journal，2010，42(1)：136-149.

[13] TIAN X H，DAI S P，DU Z Z，et al. BigDataBench-S：an open-source scientific big data benchmark suite[C]// Proceedings of IEEE International Parallel and Distributed Processing Symposium. Lake Buena Vista，USA：[s. n.]，2017：1068-1077.

[14] 屠雪真，屠要峰，陈小强. 一种优化的Key-Value型NoSQL系统[J]. 计算机工程，2019，45(6)：52-59.
TU X Z，TU Y F，CHEN X Q. An optimized Key-Value NoSQL system[J]. Computer Engineering，2019，45(6)：52-59.（in Chinese）

[15] BAO X Q，LIU L，XIAO N，et al. Policy-driven configuration management for NoSQL[C]//Proceedings of ACM International Conference on Cloud Computing. New York，USA：ACM Press，2015：245-252.

[16] JAMSHIDI P，CASALE G. An uncertainty-aware approach to optimal configuration of stream processing systems[C]// Proceedings of the 24th IEEE International Symposium on Modeling，Analysis and Simulation of Computer and Telecommunication Systems. Washington D. C.，USA：IEEE Press，2016：39-48.

（下转第217页）

（上接第211页）

[17] CHEN H F, JIANG G F, ZHANG H, et al. Boosting the performance of computing systems through adaptive configuration tuning[C]//Proceedings of the 2009 ACM Symposium on Applied Computing. New York, USA: ACM Press, 2009: 1045-1049.

[18] QI C X, BI Y M, LI Y. Improved BP neural network algorithm model based on chaos genetic algorithm[C]// Proceedings of the 3rd IEEE International Conference on Control Science and Systems Engineering. Washington D. C. , USA: IEEE Press, 2017: 679-682.

[19] XIONG W, BEI Z D, XU C Z, et al. ATH: auto-tuning HBase's configuration via ensemble learning[J]. IEEE Access, 2017, 5: 13157-13170.

[20] BAO L, LIU X, XU Z H, et al. AutoConfig: automatic configuration tuning for distributed message systems[C]// Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering. Montpellier, France: [s. n.], 2018: 29-40.

[21] 黄健宏. Redis设计与实现[M]. 北京: 机械工业出版社, 2014.
HUANG J H. The design and implementation of Redis [M]. Beijing: China Machine Press, 2014. (in Chinese)

[22] Intel. Intel DPDK: programmers guide[EB/OL]. [2020-03-10]. http://dpdk. org/doc/guides/prog_guide.

[23] ABUBAKAR Y, ADEVI T S, AUTA I G. Performance evaluation of NoSQL systems using YCSB in a resource austere environment[J]. International Journal of Applied Information Systems, 2014, 7(8): 23-27.

编辑　索书志