一种层次化的云操作系统性能诊断方法

袁 月

(中国人民大学信息学院 北京 100872)

摘 要 近年来,很多研究者致力于开发自动的性能诊断工具来应对大规模高负荷的分布式环境。云操作系统是云用户与云资源的中间层,诊断并解决云操作系统响应过慢的问题有助于优化云计算系统的性能,在大规模且复杂的分布式云计算环境下,分析云操作系统的任务执行性能具有挑战性。在此背景下,文中提出了一种基于日志的云操作系统性能诊断方法,目的是为指定类别的云操作系统任务找到其处理过慢的原因,为性能优化提供线索。该方法结合云操作系统的实现原理,从云操作系统所产生的海量日志中分离和提取每个系统执行任务相关的日志,抽取关键信息,从而构建层次化的性能描述模型,并将分析粒度逐层细化到函数执行的粒度。通过这种方法,能够找到系统任务执行过慢的主要因素,辅助定位引发性能异常的根源,无需修改源代码或借助源代码分析。以云操作系统 OpenStack 为原型系统,搭建云计算环境,并进行大规模并发模拟实验。实验结果表明,文中所提出的诊断方法能为系统性能优化提供有效线索,显著提高系统性能,例如,云资源调度过程的耗时可以从分钟级减少到秒级。

关键词 云计算,基础设施即服务,系统性能诊断,日志分析

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2019.03.047

Hierarchical Performance Diagnosis Method for Cloud Operating System

YUAN Yue

(School of Information, Renmin University of China, Beijing 100872, China)

Abstract Recently, quite some researchers aim to develop automatic performance diagnostic tools for dealing with the large-scale and high-load distributed environment. Cloud operating system is the middle layer between cloud user and cloud resource, and diagnosing and settling the problem of slow response of cloud operating system is helpful for optimizing the performance of cloud computing system. It is a challenging job to analyze the performance of executing task in large-scale and complex distributed cloud computing environment. In light of this, this paper proposed a log-based performance diagnosis method for cloud operating system to find out the reason for low execution speed of appointed tasks and provide clues for performance optimization. This method combines the implementation principal of cloud operating system, separates and extracts relevant logs of each executing tasks from the massive logs generated by cloud operating system, and extracts key information, so as to construct hierarchical performance description model and refine the analysis granularity to function executed granularity layer by layer. Finally, through using this method, the main factor of low execution speed can be gotten, which can assist to locate the source of abnormal performance, and it doesn't need to modify the source code and use the source code to conduct analysis. This paper utilized the OpenStack as prototype system, created the cloud computing environment, and conducted large-scale concurrent simulation experiment. The experimental results demonstrate that the proposed method can provide efficient clues for optimizing system performance and improve the performance obviously, e.g. the consumed time of cloud resource scheduling can be reduced from minute level to second level.

Keywords Cloud computing, IaaS, System performance diagnosis, Log analysis

1 引言

利用云计算技术能够合理整合并充分利用各处空闲异构的物理资源,从而以较低的成本投入获取计算能力及存储容量。随着云计算的发展与成熟,不仅公有云逐渐被广泛应用,私有云也逐渐普及,不少企业、学校、政府机构都迫切希望构建私有云,以在满足业务需求的同时,既降低成本开销,又减少使用公有云带来的安全隐患。越来越多的组织利用开源云操作系统(如 OpenStack[1]),按照自身的业务需求对云操作系统进行进一步开发,并将其投入生产使用,但对原本复杂的

到稿日期:2018-09-30 返修日期:2018-12-28 本文受国家自然科学基金面上项目(61472429)资助。

袁 月(1991—),女,博士生,主要研究方向为云计算、信息安全,E-mail:yyuanyuee@163.com(通信作者)。

云操作系统进行二次开发,可能引发新的系统问题,给系统异常诊断带来困难。与此同时,随着云规模的扩大以及云用户的增多,可能出现云操作系统任务处理过慢,从而导致云用户服务请求响应延迟的情况,此时通常很难定位此类性能问题的原因,而这种性能问题会直接影响用户访问云资源,甚至导致云用户无法获取云资源。因此,需要一种有效的性能诊断方法,来及时发现云操作系统在任务执行过程中的关键问题,以帮助修复和优化云计算环境。

云是基于服务的系统,云用户服务请求的响应时间直接反映了系统性能[2],因此,通过跟踪处理用户请求的云操作系统任务,分析相关关键数据,有助于诊断云操作系统的性能。目前,有很多插桩追踪工具[3-5]通过分析端对端的请求跟踪数据来定位系统性能出现异常的原因。例如,最近 Facebook 提出了追踪工具 Canopy[5],其能够帮助诊断大规模跨服务组件的系统性能问题。但这些工具通常涉及系统源代码的修改。

另一方面,日志直接反映系统的运行情况,常被用于系统行为分析。近年来,有很多研究工作[6-11]以及性能异常分析工具[12-16]通过对日志数据进行挖掘来检测和诊断大规模分布式系统中的异常。例如,文献[7-8]利用标识符关联的方法对日志进行分组,从日志中抽取出与系统任务执行对应的日志事件序列,从而对系统任务执行中的问题进行分析。文献[11]通过分析 API 序列来对云环境进行审计,其中涉及云操作系统任务 API 与日志事件之间映射关系的建立,如利用一个日志事件来标识云环境中创建虚拟机的任务。一些文献或是将日志分析与源代码静态分析的结果相结合[13,16],或是利用机器学习的方法[14-15]来分析系统中的性能异常。但目前仍然缺乏一种简单有效的云操作系统性能诊断方法。

本文提出一种半自动化的层次化的云操作系统性能诊断方法,其目标是为指定类别的云操作系统任务找出执行过慢的关键因素。所提方法将云操作系统日志作为唯一的数据来源,从日志中挖掘出能够描述系统任务执行耗时情况的信息,建立层次化性能描述模型,通过人工干预的方式,逐层细化诊断粒度,最终定位出耗时过长的函数调用,为性能优化提供线索。

与目前已有方法相比,本文提出的诊断方法基于系统现有日志来分析云环境中的性能问题,不需要修改系统源代码或借助源代码进行分析,也不涉及监督学习的过程,目的是为指定类别的云操作系统任务找到其处理过慢的原因。不同于有的研究[12]挖掘日志中涉及的各种对象关系,本文结合云操作系统的实现特点,利用日志事件序列及相关信息来建模描述系统任务的执行,并利用日志事件来标识不同的系统任务,能够针对某一类云操作系统任务,定位到函数执行级别的性能异常的原因。另外,自动化日志分析的第一步通常涉及日志解析,将非结构化的日志转化为结构化的日志事件,与源代码建立映射关系。目前已有很多工具[17-18]能够自动进行日志解析,这些日志解析工具均可以应用在本文所提方法中。

我们以云操作系统 Openstack 为原型系统,搭建云计算环境作为实验环境,以创建虚拟机任务为例进行详细分析。实验结果表明,在大量并发创建虚拟机的情况下,利用本文提出的诊断方法,可以针对耗时较长的创建任务给出诊断结果,能够为系统性能优化提供线索,有效地减少创建耗时。

本文第 2 节以 OpenStack 为代表介绍云操作系统的相关背景;第 3 节阐述云操作系统的性能诊断方案;第 4 节给出实现及实验评估结果;最后总结全文。

2 云操作系统的相关背景

在阐述方法设计前,以 OpenStack 为例,对云操作系统作简要介绍。OpenStack 的架构如图 1 所示。

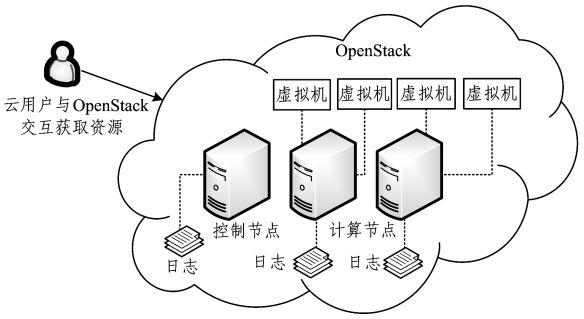


图 1 OpenStack 架构

Fig. 1 Typical architecture of OpenStack

OpenStack 是基于 Python 语言实现的开源分布式云操作系统,它由多个服务模块构成,包括计算服务(Nova)、镜像服务(Glance)、网络服务(Neutron)、认证服务(Keystone)等,通过各个服务之间的合作交互来共同完成云用户的请求。系统中包含多个节点,通常分为控制节点、存储节点、网络节点、计算节点四大类,各类均可以通过自身需求融合为同一节点。每个服务由多个服务进程组成,同一个节点上可能运行多个不同的服务进程。

另外,OpenStack 还依赖其他基础服务来完成用户请求,其中涉及两大通信机制:远端程序调用(RPC)使用高级消息队列协议(AMQP),服务器网关接口(WSGI)使用 REST 框架。每个服务内部的组件之间通过 RPC 进行通信,例如,在计算服务中,负责接收用户请求的服务进程 nova-api 与负责资源调度的服务进程 nova-scheduler 之间,nova-api 接收用户请求之后,如果需要进行资源调度,则通过 RPC,交由 nova-scheduler 完成调度任务。服务与服务之间则通过 WSGI 进行通信,例如,计算服务与网络服务通过 http 请求进行通信,用户可以使用命令行(CLI)或 Web 浏览器通过 REST 请求访问各个服务的 API 接口。另外,OpenStack 利用 SQLAlchemy 库与数据库后端进行通信来管理数据信息,如 MySQL,SQLite 等;OpenStack 与虚拟机监控器的交互则是通过抽象虚拟化驱动完成,OpenStack 的驱动接口基于虚拟机监控器 API 实现,如 Libvirt,Xen API;OpenStack 各服务通过调用 eventlet 和 greenlet 库实现多线程异步机制。

在 OpenStack 系统的运行任务处理过程中,每个任务都可能涉及上述位于不同服务节点的多个服务进程。如图 1 所示,分布在不同节点上的不同服务进程有各自的日志文件,这些日志文件记录各服务进程的运行状态,而有助于定位任务执行耗时过长的原因的线索则隐藏在各节点和各个组件日志

信息中。本文通过挖掘 OpenStack 所有组件的日志,并与任务类型关联,来帮助更全面且细粒度地分析整个系统的运行状况。

3　性能诊断方法的设计

3.1　整体架构

本文提出的性能诊断方法中提到的性能是指云操作系统执行任务的耗时,诊断的目的是找出导致指定系统任务耗时过长的函数调用,为性能优化提供线索。该诊断方法主要分为 3 个步骤(见图 2):数据预处理、模型构建和性能诊断。

数据预处理:从云环境各服务节点中收集云操作系统日志;然后,一方面,从日志中提取标志任务执行的日志事件以及相关的代表任务执行的请求标识符,另一方面,从这些日志中分离出代表任务执行的日志事件序列,并提取相关时间戳、来源等关键信息。

模型构建:为了系统化地描述任务执行的耗时情况,构建层次化性能描述模型,从任务执行、进程和代表函数调用的日志事件这 3 个层面进行描述,且描述的粒度逐层扩展、细化。

性能诊断:基于所构建的层次化性能描述模型,以人工干预的方式细化并诊断某类任务的性能异常,最终给出可能导致执行该类任务耗时过长的函数调用,为进一步的性能优化提供线索。

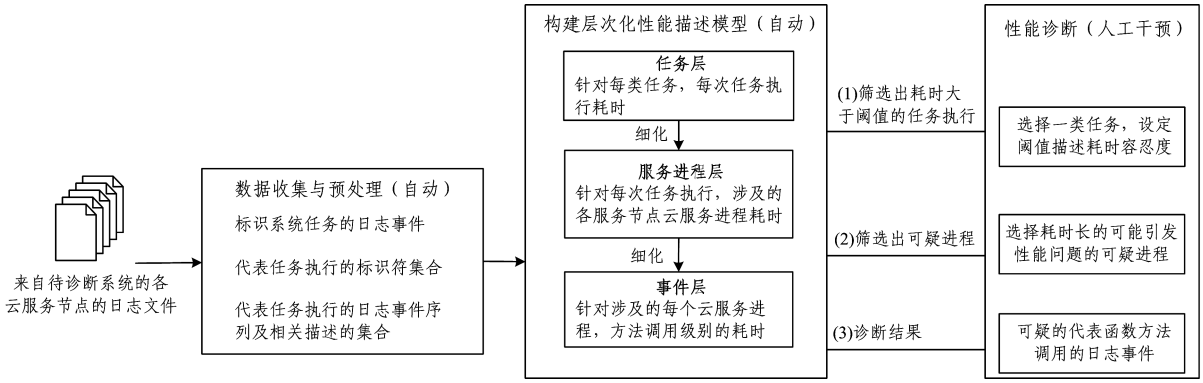


图 2　诊断方案

Fig. 2　Diagnostic approach

3.2　日志数据的收集与预处理

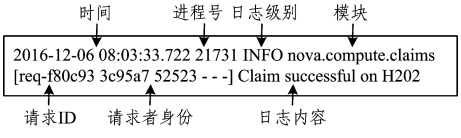
本节首先结合云操作系统的实现特点,介绍日志收集过程和预处理中涉及的日志信息分组的方法;然后详述日志预处理中涉及的日志事件提取的方法;最后总体阐述日志数据预处理的过程。

3.2.1　日志收集与日志信息分组

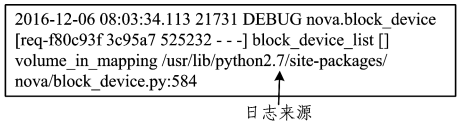
云操作系统中每个服务组成部分由多个服务进程构成,如 OpenStack 中的 Nova 服务包含 nova-api,nova-scheduler 等服务进程;按照部署规划的不同,各个组件可能分布在不同的节点上,如 nova-api 一般在控制节点上。nova-compute 一般在计算节点上。通常,在默认情况下,每个服务进程有自己独立的日志文件,用以记录该进程运行时的情况。本文将各个节点上各服务进程对应的日志文件收集起来作为数据源。

日志通常是开指发人员在系统调试时插入源代码中的语句,日志信息对分析云环境中的操作有重要的作用。系统日志,特别是对于以提供服务为主要任务的系统,记录了系统运行时的行为以及关联的上下文。以 OpenStack 为例,云操作系统处理用户请求的过程会涉及上下文参数,上下文中包含请求标识符,可以通过请求标识符关联出系统处理同一任务产生的日志,其默认系统日志在 Python 原有的日志模块的基础上做了一层封装。与每个请求处理相关的日志中都含有请求标识符,用以标识该请求。

如图 3 所示,本文涉及的日志属性为时间戳、请求标识符、日志内容。时间戳可以确定事件发生的时间和顺序,请求标识符可以关联同一个任务执行产生的主要日志,日志内容可以辅助判断当前的任务处理位置。



(a)其他级别的日志



(b)DEBUG 级别的日志

图 3　OpenStack 日志格式实例

Fig. 3　Simplified examples of logs in OpenStack

3.2.2　日志事件提取

通过日志解析工具,可以将原始日志转化为结构化的日志事件。本文采用的是一种基于最长公共子串的日志解析方法[18]。

一条日志由源代码中的日志输出语句产生。一条日志输出语句中通常包含格式化字符串以及输出参数变量,因此,由同一条日志输出语句产生的日志具有相同的子字符串,与日志输出语句中的字符串对应。而日志解析,就是找出由同一条日志输出语句产生的日志,并抽取出它们的一般模式作为日志事件,该模式通常与日志输出语句具有映射关系。例如,一条日志“2017-11-15 09:59:14.385 req-342 Security group member updated u'e0217e41-926e-47ed-a”由源代码中的日志输出语句“LOG.debug('Security group member updated %s',

security groups)”产生，解析后会得到日志事件“Security group member updated . * ”。一个日志事件的出现，代表日志输出语句的执行，也代表包含该日志输出语句的函数的调用。

3.2.3 日志数据预处理

给定 n 条日志记录 $M=\langle l_1,l_2,\cdots,l_n\rangle$，首先提取日志中的请求标识符，以及与每个请求标识符相关的首条日志，然后将这些日志解析成为日志事件，从而得到：

$$R=\{(r_i,e_i)\mid 1\leqslant i\leqslant p\}$$

其中，r_i 是从日志中提取的请求标识符，e_i 是由相关首条日志转化得到的日志事件且能够标识云环境中诸如创建虚拟机任务的云操作系统任务，p 是日志中出现的请求标识符的总数。

接着，将原始日志按照请求标识符进行分组，并将每条日志 l_i 解析后转化为一个五元组：

$$T(l_i)=(t_i,n_i,h_i,c_i,e_i)$$

其中，t_i 是时间戳；n_i 是该日志在来源日志文件中的所在行数；h_i 是来源日志所在的服务器节点；c_i 是产生该日志文件的服务进程，由日志文件名标识；e_i 是解析后的日志事件。从而可以得到分组后的集合：

$$G=\{(r_i,\langle T(l_{i_1}),T(l_{i_2}),\cdots,T(l_{i_{end}})\rangle)\mid 1\leqslant i\leqslant p\}$$

其中，r_i 是请求标识符，l_{ij} 是与该请求标识符相关的日志，p 是日志中出现的请求标识符的总数。G 中的每个元素 $T(l_{ij})$ 为五元组组成的序列，该序列按照每个元组中的时间戳 t_i 和行号 n_i 进行排序，该序列描述的是一个任务的一次执行情况。

3.3 层次化性能描述模型的构建

该模型分为任务层、进程层和事件层，对云环境中的任务执行情况进行描述，逐层细化云环境中任务执行的性能描述粒度。

3.3.1 任务层

基于预处理后得到的数据集 G，针对由请求标识符标识的每次任务执行，计算开始时间和结束时间的差值，然后结合预处理得到的 R，将请求标识符进行分组。至此，可以得到任务层的性能描述：

$$D_{\text{task}}=\{e_i:\{(r_{ij},s_{ij})\mid 1\leqslant j\leqslant q_i\}\}$$

其中，e_i 是标识云操作系统任务的日志事件；r_{ij} 是请求标识符标识该任务的一次执行；s_{ij} 是计算得到的时间差，表示一次任务执行在日志中记录的起止时间；q_i 是一类任务总共的执行次数。

3.3.2 进程层

针对预处理得到的 G 中的每个元素，按照每个五元组中的服务节点 h_i 和服务进程 c_i 对该序列进行分组，得到若干子序列，然后针对每个子序列，计算首末元组的时间差，从而得到进程层的性能描述：

$$D_{\text{daemon}}=\{r_i:\{(h_{ij},c_{ij},s_{ij})\mid 1\leqslant j\leqslant k_i\}\}$$

其中，r_i 是请求标识符；h_{ij} 和 c_{ij} 是相关的服务节点和服务进程；s_{ij} 是计算得到的时间差，表示该服务进程执行该任务中的一个子任务所花费的时间；k_i 是该次任务执行涉及的所有分布式服务进程的总数。

3.3.3 事件层

对于上一步基于 G 分组得到的子序列，每个子序列中的每个元组代表的是所含日志事件的发生，表示对应的日志输出语句的调用，也表示包含该日志输出语句的函数的调用。这一步主要累计日志事件与其上一个相邻日志事件之间的时间差值，该值能够一定程度地描述相关函数的执行性能。具体而言，对于每个子序列，首先为每个元组计算一个中介值，即该元组与前一个元组的时间差值，然后针对每个在该序列中出现的日志事件，累加相关元组对应的中介值，从而得到事件层的性能描述：

$$D_{\text{event}}=\{(r_i,h_{ij},c_{ij}):\{(e_{ijm},s_{ijm})\mid 1\leqslant m\leqslant v_{ij}\}\}$$

其中，r_i 是请求标识符，h_{ij} 和 c_{ij} 是相关的服务节点和服务进程，e_{ijm} 是日志事件(与源代码中的日志输出语句具有对应关系)，s_{ijm} 是计算得到的性能描述值，v_{ij} 是该子任务涉及的日志事件总数。

3.4 性能诊断

层次化性能描述模型的任务层、进程层和事件层构建完成后，可针对云操作系统中的执行性能，逐层递进地进行细粒度的分析。

3.4.1 任务层性能诊断

每类云操作系统任务(如在云环境中创建虚拟机的任务)由日志事件标识，每次任务的执行由请求标识符标识。3.3.1 节中得到的性能描述 D_{task} 描述的是任务与该任务实例化执行性能之间的映射关系，因此可以直接基于此数据集，选择需要诊断的任务类型，然后针对每类需要诊断的任务，通过比较已经计算得到的每个执行任务在日志中记录的起止时间差与给定阈值，找出异常的执行任务。其中的阈值人为给出，表示的是对该类任务执行的耗时容忍度。

3.4.2 进程层性能诊断

针对每类任务，基于 3.4.1 节的输出结果，可以得到性能异常的执行任务集合。3.3.2 节得到的 D_{daemon} 描述的是每个执行任务与相关服务进程执行性能之间的映射关系，以此为基础，为了给每个异常执行任务提供诊断结果，分别从服务节点和服务进程两个方面进行考虑。针对某一类任务的执行过程所涉及的每个服务节点，在 D_{daemon} 中以服务节点为主键聚集相关的时间值，并计算平均值、最大值、最小值作为统计结果。类似地，针对服务进程，以服务进程名为主键，计算相应的统计结果，从而可以找出压力大的服务节点，并挑选出可疑的服务进程，帮助分析者进行进一步的诊断和修复。

3.4.3 事件层性能诊断

针对每个挑选出来的可能导致性能异常的服务进程，基于得到的相关事件层的性能描述，即 3.3.3 节得到的 D_{event}，对所涉及的日志事件按照对应性能描述值的大小进行排序，找出导致性能异常的可疑事件，从而针对诊断得到的日志事件，进一步得到相关的源码中的函数调用，并采取相应的性能优化措施。

4 实现与实验验证

本节首先介绍应用本文提出的性能诊断方法实现云操作

系统性能分析工具的过程,然后通过实验来验证该工具的有效性。

4.1　实现

根据本文所提出的性能诊断方法,用 Python 语言实现了一个云操作系统性能分析工具。

为了不给云环境带来额外的负担,先将系统中的日志文件统一收集到负责日志处理的服务器上,再进行进一步分析,而没有采取直接在云服务器节点上对原始日志进行处理的方式。

在日志数据预处理阶段,通过顺序处理每个日志文件来提取标志任务执行的日志事件以及相关的代表任务执行的请求标识符。同时,通过并发处理的方式从这些日志中分离出代表任务执行的日志事件序列,并提取相关时间戳、来源等关键信息。最后,将这些信息统一汇总。

在模型构建阶段,按照模型中任务层、进程层和事件层的顺序逐层进行数据处理;在性能诊断阶段,通过人工干预的方式按顺序逐步细化诊断结果,直到最终定位出性能问题的根源。

4.2　实验验证

实验环境:以 OpenStack 为原型构建实验环境,其中包括 1 个控制节点服、50 个计算节点、3 个存储节点(以 Ceph[19] 作为存储后端)。所有服务组件的日志级别均设置为 DEBUG 级别。

实验过程:以创建虚拟机请求为例进行实验,实验时,在实验环境中执行大量的虚拟机创建操作,同时将删除虚拟机、迁移虚拟机的操作作为干扰操作,然后利用本文所实现的性能分析工具获取各服务器节点上的 OpenStack 系统日志[20],提取出与创建任务相关的关键信息,定位 OpenStack 执行虚拟机创建任务中的性能瓶颈。

4.2.1　计算组件核心的性能问题诊断

实验中共涉及 5 组创建虚拟机的操作,每组并发创建 50 台虚拟机。通过任务层的分析,定位出其中 3 组操作请求各自耗费的总时间均在 3 min 以上。通过进程层的进一步分析发现,这 3 组操作请求均在计算组件核心服务进程 nova-compute 上耗时最长。进一步,可以得到 nova-compute 相关的事件层的分析结果,其涉及 81 个日志事件,其中 78 个日志事件对应的计算时间值均小于 10 s,剩下 3 个事件的结果如表 1 所列。可以看出,准备镜像资源和启动虚拟机这两个过程的耗时较长,尤其是准备镜像资源。

表 1　创建虚拟机实验的诊断结果展示

Table 1　Diagnosis results of VM creation tasks

事件	时间值/s	说明
fetch_func_sync	346.187	创建镜像资源
instance is running	48.575	Hypervisor 启动虚拟机
ensure console log	19.987	调整虚拟机镜像后准备虚拟机日志文件

根据此线索,进一步分析发现,由于 Ceph 配置不当,存储副本配置数目过多,导致其处理大量并发创建镜像请求时发生任务阻塞,调整副本数配置后,创建虚拟机的性能有了极大的提升,耗时有所减少。

4.2.2　调度器中的性能优化

对 Ceph 进行优化后,在进一步大量创建虚拟机的过程中发现,再次出现瓶颈问题(调度时间超过 1 min,OpenStack 默认的配置等待调度时间为 1 min)。通过诊断分析后,根据诊断工具给出的线索,对系统进行优化。创建虚拟机实验的结果如图 4 所示。其中,a 为 begin_end,b 为 filtered,c 为 AvailabilityZoneFilter,d 为 ComputeCapabilitiesFilter,e 为 ComputeFilter,f 为 ImagePropertiesFilter,g 为 RamFilter,h 为 RetryFilter,i 为 ServerGroup-AffinityFilter,j 为 ServerGroupAntiAffinityFilter,k 为 weighed,l 为 select。

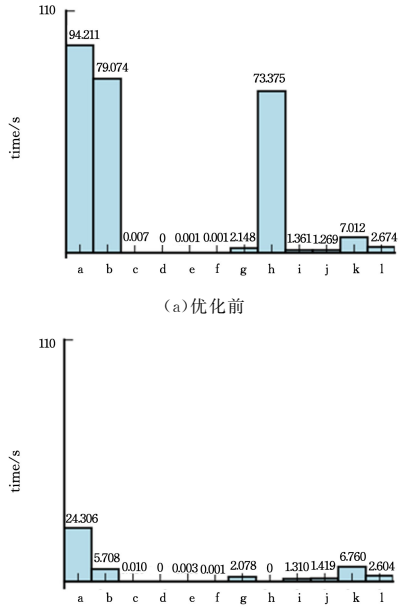


图 4　创建虚拟机实验的结果展示

Fig. 4　Diagnosis results of VM creation tasks

在性能优化前后,分别执行相同的创建操作,即一个请求中同时创建 1500 台虚拟机。诊断分析结果如图 4(a)所示,其中 begin_end 指 nova-scheduler 服务进程所花费的总时间,其他的是由日志事件标识的该进程执行的各个阶段。优化前,进一步通过进程层分析结果,如图 4(b)所示。可以看出,Retryfilter 重试过滤器的耗时长,引起过滤总时间(filtered)变长,从而导致调度总时间(begin_end)变长,造成调度器出现性能问题。查看配置文件,调度器未配置重试机制,Retryfilter 可以去掉,因此,考虑通过在配置文件中去掉 Retryfilter 来提高调度器性能。修改配置文件后,再次进行测试,得到如图 4(b)所示的结果,可以看出,调度总时间减少到秒级,性能得到了大幅提高。

结束语　本文提出的诊断方法可以有效地为定位云操作系统任务处理过慢的原因提供线索,辅助优化系统性能,以进一步加快云操作系统任务的处理过程。本文提出的诊断方法可以有效地应用在实际场景中,例如,当云用户(包括云管理员、开发者)在执行某一项操作的过程中发现系统响应速度很慢时,可以借助该方法进行分析,找到造成该性能瓶颈的根源,从而进行针对性修复和优化。

与现有大部分性能分析工具相比，本文提出的性能诊断方法是基于现有日志的，不更改原系统的代码，在无法获取系统源代码的情况下仍然可以使用。层次化粒度逐步细化的方法能够为指定的系统任务提供诊断结果，比直接从日志中抽取所有任务的执行路径并合并分析更具针对性。

实验结果表明，本文所提出的性能诊断方法能够有效地在大规模云环境中针对指定系统任务给出分析结果，运维开发人员可以根据性能分析结果有效定位需要进行性能优化的代码模块，从而提高云计算系统任务的处理效率。然而，本文所提方法仍然依赖于一些人工干预，在未来的工作中，我们会继续深入，研究力求提出一种更加自动化的性能诊断工具。

参 考 文 献

[1] OpenStack. OpenStack open source cloud computing software [EB/OL]. [2018-06-05]. http://www.openstack.org.

[2] MI H B, WANG H M, ZHOU Y F, et al. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2013, 24(6): 1245-1255.

[3] SAMBASIVAN R R, ZHENG A X, ROSA M D, et al. Diagnosing performance changes by comparing request flows[C]// Proceedings of USENIX Conference on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2011: 43-56.

[4] SIGELMAN B H, BARROSO L A, BURROWS M, et al. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure [R]. Google Technical Report, 2010.

[5] KALDOR J, MACE J, BEJDA M, et al. Canopy: An End-to-End Performance Tracing And Analysis System[C]// Proceedings of ACM Symposium on Operating Systems Principles. New York: ACM Press, 2017: 34-50.

[6] NANDI A, MANDAL A, ATREJA S, et al. Anomaly detection using program control flow graph mining from execution logs [C]// Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM Press, 2016: 215-224.

[7] SHANG WY, JIANG Z M, HEMMATI H, et al. Assisting developers of big data analytics applications when deploying on hadoop clouds[C]// Proceedings of International Conference on Software Engineering. New York: IEEE Press, 2013: 402-411.

[8] LIN Q W, ZHANG H Y, LOU J G, et al. Log clustering based problem identification for online service systems[C]// Proceedings of International Conference on Software Engineering Companion. New York: IEEE Press, 2016: 102-111.

[9] HE S L, ZHU J M, HE P J, et al. Experience report: System log analysis for anomaly detection[C]// Proceedings of IEEE International Symposium on Software Reliability Engineering. New York: IEEE Press, 2016: 207-218.

[10] WU F, ANCHURI P, LI Z H. Structural event detection from log messages[C]// Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM Press, 2017: 1175-1184.

[11] MAJUMDAR S, JARRAYA Y, OQAILY M, et al. Leaps: Learning-based proactive security auditing for clouds[C]// Proceedings of European Symposium on Research in Computer Security. Berlin: Springer, 2017: 265-285.

[12] ZHAO X, RODRIGUES K, LUO Y, et al. Non-intrusive performance profiling for entire software stacks based on the flow reconstruction principle[C]// Proceedings of USENIX Conference on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2016: 603-618.

[13] ZHAO X, ZHANG Y L, LION D, et al. Lprof: A non-intrusive request flow profiler for distributed systems[C]// Proceedings of USENIX Conference on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2014: 629-644.

[14] ROY S, KONIG A C, DVORKIN I, et al. Perfaugur: Robust diagnostics for performance anomalies in cloud services[C]// Proceedings of IEEE International Conference on Data Engineering. New York: IEEE Press, 2015: 1167-1178.

[15] NAGARAJ K, KILLIAN C, NEVILLE J. Structured comparative analysis of systems logs to diagnose performance problems [C]// Proceedings of USENIX Symposium on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2012: 353-366.

[16] YUAN D, MAI H, XIONG W, et al. SherLog: error diagnosis by connecting clues from run-time logs[C]// Proceedings of ACM International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2010: 143-154.

[17] HE P J, ZHU J M, HE S L, et al. Towards automated log parsing for large-scale log data analysis[J]. IEEE Transactions on Dependable and Secure Computing, 2018, 15(6): 931-944.

[18] DU M, LI F. Spell: Streaming parsing of system event logs[C]// Proceedings of IEEE International Conference on Data Mining. New York: IEEE Press, 2016: 859-864.

[19] WEIL S A, BRANDT S A, MILLER E L, et al. Ceph: A scalable, high-performance distributed file system[C]// Proceedings of USENIX Conference on Symposium on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2006: 307-320.

[20] LIU J L, CHENG C Y, CHEN Z, et al. Research on Cloud Data Management Model Based K-Means and Gridding Clustering [J]. Journal of Chongqing University of Technology (Natural Science), 2017, 31(9): 125-130. (in Chinese)
刘加伶，程春游，陈庄，等. 基于 K-Means 和网格化聚类的云数据管理模型研究[J]. 重庆理工大学学报(自然科学)，2017，31(9)：125-130.