DOI:10.7544/issn1000-1239.2016.********
卷(期):起止页,年

面向云计算平台的周期性任务调度方法

张鹏[1,2] 李焱[3] 窦凤虎[1,2] 檀国林[1,2] 刘庆云[1,2]
[1]中国科学院信息工程研究所，北京 100091
[2]信息内容安全技术国家工程实验室，北京 100091
[3]国家计算机网络应急技术处理协调中心，北京 100029
(pengzhang@iie.ac.cn)

An Approach to Periodic Task Scheduling Towards Cloud Computing

Zhang Peng[1] Li Yan[3] Dou Fenghu[1,2] Tan Guolin[1,2] and Liu Qingyun[1,2]
[1](Institute of Information Engineering, Chinese Academy of Science, Beijing 100091)
[2](National Engineering Laboratory for Information Security Technologies, Beijing 100091)
[3](National Computer Network Emergency Response and Coordination Center, Beijing 100029)

Abstract As cloud computing enables ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction, more and more enterprises build their applications based on the model, however, both efficient resource management and effective task scheduling require in-depth analysis of hosted applications and adoption of corresponding optimization techniques, for this purpose, based on the periodic feature of time-spatial correlate task, this paper presents a periodic task-oriented scheduling framework to achieve the goal of "one scheduling, many execution", and makes research on the scheduling of different tasks on the same resource node, and proposes a RM-FFDU4PT scheduling algorithm, which can meet the execution demand of batch periodic tasks with minimal resources. Experimental results on a real periodic task set show that the proposed framework can reduce the scheduling times to less than 10%, which greatly reduces the load on the platform.

Key words cloud computing; task scheduling; periodic feature

摘要 云计算通过提供动态、可伸缩的计算资源，以满足不同层次、多样化的计算需求，由于可以节省应用系统部署和运维的成本，越来越多的企业采用该模式构建应用系统，然而，实现云计算平台合理的资源管理与任务调度需要深入分析平台所承载应用的特征并采取有针对性的优化方法,为此本文针对任务的周期性特征提出一种面向周期性任务的调度框架，实现任务的“一次调度、多次执行”，其中通过 RM-FFDU4PT 算法，能够用尽量少的资源满足批量周期性任务的执行需求。在实际周期性任务负载集上的实验表明，该框架能够将任务调度次数降低至原来的 10%以下，极大降低平台调度负载。

关键词 云计算；任务调度；周期性特征

中图法分类号 TP391

收稿日期：yyyy-mm-dd Date 六号 修回日期：yyyy-mm-dd Date 六号
基金项目：国家自然课基金项目(61402464,61300206)；
This work was supported by the National Natural Science Foundation of China(61402464,61300206)

DOI:10.7544/issn1000-1239.2016.********
卷(期):起止页,年

随着云计算技术的不断发展，越来越多的应用走向云端，然而，这些应用个性化特征也给云计算平台的任务调度和资源管理带来的巨大的挑战。以舆情信息获取任务为例：在时间维度上，不同时刻对同一采集源的采集结果可能有所不同，所以通常要求对重点采集源的采集频率达每 5 分钟一次或是更高，然而，现有的云计算平台并未考虑这类任务的时序特征，不管任务是否为重复执行，一律执行一次、调度一次。这样至少从以下几个方面影响了运行效率：

- 增加了主节点调度压力。每一个周期性任务的每一次运行都需要主节点进行调度，加大了该节点的负载，使得该节点极易成为系统的瓶颈；
- 增加了数据中心网络压力。跨域数据中心不同域间的带宽资源十分宝贵，频繁的任务下发容易造成网络拥塞；
- 相对降低了资源节点的任务执行效率。资源节点在每次执行任务前都需要判断执行代码在本地是否存在并视情况下载任务代码，而有些任务的实际执行时间在分钟级别。频繁的执行代码存在性判断甚至代码下载操作占用了资源节点的时间，降低了任务实际执行效率，客观上加剧了稀缺资源的紧张态势。

下面给出一个案例分析。有一个 DNS 服务监测业务 A 需要每 10 分钟 1 次监测某 DNS 服务器面向广州电信用户的域名解析服务运行情况，持续监测时间为 1 个月，1 次监测服务所需要的时间为 3 分钟。在一个月内该业务共需要向云计算平台下发 4320 个任务。上述业务所对应的任务可描述为 t_a={558,广州,{2,1,1,{CT,}},0,3,10,1m }，558 为任务的 id，广州表示任务放置位置约束，接下来的 2、1、1 分别表示要求消耗 2 个单元的 CPU、1 个单元的内存、1 个单元的硬盘空间（平台已将上述资源单元化），CT 表示需要资源节点接入互联网的基础运营商为中国电信，该任务对网络接入方式和带宽无要求，0 表示该任务可立即执行，3 表示任务单次执行的时长为 3 分钟，10 表示运行周期长为 10 分钟，1m 表示该任务需持续间断运行 1 个月。

有了上述描述信息之后，通过感知任务的时序特征，平台可以做到只调度 1 次，即为该任务分配满足其执行需求的资源节点。之后，该任务的每次执行都可由该资源节点自行组织完成，可表示为 t_{aj}，即任务 t_a 的实例。

然而，从资源利用率的角度来看，上述优化是不够的，该资源节点超过 40%的时间是空闲的。因此，需在满足时间约束的条件下将尽量多的任务放在同一资源节点执行。

假设在平台还运行着某大型网络软件系统效果嗅探业务 B，需每 10 分钟 1 次从全网嗅探关键信息，持续时间为 1 个月。其中包含两个任务 t_b 和 t_c，单次执行时间分别为：3 分钟和 5 分钟。表示如下：

t_b= {561,广州,{1,1,1,{CT, }},0,3,10,1m }

t_c= {562,广州,{1,1,1,{CT,}},0,5,10,1m }

t_b 和 t_c 除了单次运行时长不同外其余基本上一致，但与任务 t_a 相比，所需资源一样，即它们可以运行在同一台资源节点上。t_a，t_b 和 t_c 的周期大小一致都为 10 分钟，但三者单次执行耗时总和为 11 分钟，实际上这三个任务无法在同一资源节点上同时不间断执行（如下图 1 所示）。图中任务简单表示为（周期长，可执行时间点，单次执行时长）。

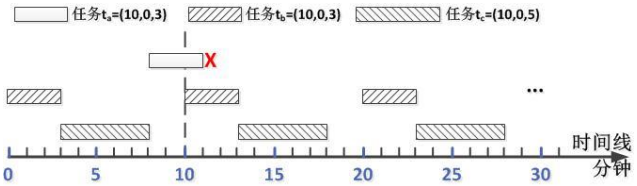

图 1. 同一资源节点上周期性任务执行实例

Fig.1 Periodic Task Scheduling in One Node

当任务较多且各任务周期性时长不一时，判断批量任务在同一资源节点上的可调度性并非易事。为此本文提出了一种面向周期性任务的调度框架，实现“一次调度、多次执行”，极大降低了平台调度负载，并且提出一种资源分配算法 RM-FFDU4PT，在满足各节点任务可调度性的前提下，使得批量任务执行所需的资源最少。

1.相关工作

现有主流云平台的任务调度架构基本上可以分成以下三类：集中式调度、分布式调度和分层调度。任务调度策略根据各个任务运行的时间点是否都发生在运行之前，可分为静态调度、动态调度和混合调度，下面详细进行介绍。

1.1 调度架构

集中式调度是最基本的调度架构，由单个集中

收稿日期：yyyy-mm-dd Date 六号 修回日期：yyyy-mm-dd Date 六号
基金项目：国家自然课基金项目(61402464,61300206)；
This work was supported by the National Natural Science Foundation of China(61402464,61300206)

DOI:10.7544/issn1000-1239.2016.********
卷(期):起止页,年

式管理器完成任务和资源的映射。MapReduce 计算框架即采用集中式调度[1]。JobTracker 运行在主节点上，负责确定作业（Job）执行计划，并为作业所包含的任务（Task）分配计算节点、监控任务的执行情况，具体可分为 Map 任务和 Reduce 任务。TaskTracker 运行在从节点上，拥有一定数量的时间片（slot）来执行所分配的任务。集中式调度架构实现较为简单，各域的资源统一管理，不用考虑资源状态不一致的问题，Alsched[2]、Quincy[3]、Paragon[4]等都属于此种调度方式。但该方式的主要缺点是可扩展性差，当云数据中心资源量或是支撑的任务量过大时，调度模块极易成为系统瓶颈。

分布式调度属于无中心的调度方式，通常无统一的资源分配模块，各任务调度器并行工作，并依据自己掌控的资源状态信息做出调度决策。Tarcil[5]是一种典型的分布式调度架构（如图 2 所示），它无中心调度器，所有的调度代理并行工作，并有一份关于所有资源（执行单元）状态的本地副本。系统中存在一个针对全局所有资源状态的 master copy，各并行代理会每隔 5-10 秒钟与之同步。为减轻 master copy 所在服务器的负载，同步时间间隔取随机值，以使各调度代理尽量不在同一时间点与之同步。在进行调度时代理会把 master copy 锁住并执行原子操作完成资源分配。每台资源节点（woker）服务器部署有本地管理程序（local monitor）以处理调度请求，并管理服务器资源的协同使用，提升执行单元的服务质量。Omega[6]、Sparrow[7]也都采用了无中心的调度方式，所有的资源分配决策都由分布式调度器来完成。

分布式调度的优点是通过调度的并行化实现任务的高吞吐调度，降低任务调度延迟，提高任务的执行效率。缺点在于因缺乏全局的控制视角，需采用复杂的方式保持不同调度器之间资源信息的一致性，加大了任务调度模块的实现难度。

分层调度介于集中式调度和分布式调度之间，结合了两种调度方式的优点，常见的为两层调度。两层调度可看作是一种分而治之的机制或者是策略下放机制：调度器仍保留一个经简化的集中式资源调度器，但具体任务相关的调度策略则下放到各个应用框架调度器来完成，这种调度器的典型代表是 Mesos[8]和 Yarn[9]。而在考虑属于同一作业(逻辑单元)的多个跨域任务的调度时，往往也会采用两级调度方式，在中心节点实现作业级调度，在各域任务调度器上实现任务级调度[10]。

集中式调度、分布式调度和分层调度等三种调度架构基本上能覆盖主流的云平台调度架构方式。然而，各云平台大都有自己的一套架构和资源管理方式，并没有形成统一的标准或者管理规范。特别是在调度架构细节上，同一类别的架构之间也存在很大的不同。以两层调度架构为例，Mesos 采用基于 Resource Offer 的 DRF 资源分配机制，而最终是否接受分配结果由 Framework 本身来决定。而 Yarn 的 RM 则基于应用的请求并结合优先级、资源的可用性等动态地为应用分配资源，决定权在于 RM 而不是应用本身。另外，Mesos 允许每个 Framework 决定采用哪种算法来调度自身的任务在所分配的资源上执行，而 Yarn 则由 RM 决定各应用所属任务的执行资源节点。总体来讲，Mesos 框架下资源分配粒度较粗，可以同时支持长时运行任务和短时运行任务，如长时 Web 服务，较适用多种计算框架并存的场景；而 Yarn 中资源分配粒度较细，直接由 RM 将资源分配至各应用所属的任务，Yarn 主要针对原有 Hadoop 中数据密集型批处理任务进行优化，既不是为长时间运行的任务而设计，也不是为满足短期交互/快速响应式任务。

然而，针对舆情信息获取这样周期性任务的资源调度，上述云平台都没有给出一个合适的调度架构和调度算法

1.2 调度策略

任务调度策略根据各个任务运行的时间点是否都发生在运行之前，可分为静态调度、动态调度和混合调度；根据任务是在一个处理机上或是多个处理机上运行，可分为单处理机和多处理机实时调度；根据任务间是否互相抢占，可以分为抢占式调度和非抢占式调度；根据调度驱动机制的不同，可分为时间驱动的调度算法、优先级驱动的调度算法、共享驱动的调度算法等[11]。不同的调度方式具有各自的优缺点，适用于不同类型实时系统及实时任务负载。下面介绍一下典型的周期性任务调度算法。

RM 调度算法由 Liu 和 Layland 于 1973 年提出[12]，它根据任务的周期大小来分配优先级，周期越小、任务的优先级越高。同时，Liu 和 Layland 证明了 RM 算法是静态优先级调度算法中最优的，即对于在任何其他静态优先级算法下可调度的任务集合，在 RM 算法下也是可调度的。

EDF 调度算法是一种动态优先级调度算法，它按照当前作业的绝对截止期为其分配优先级，作业的绝对截止期越短，其优先级别越高。EDF 调度算法中，具有最高优先级别的作业总是最先得到执行。如果当前有其他较低优先级作业正在执行，则该较低优先级作业被抢占，让位给具有最高优先级的作业执行，直至就绪队列中没有高于该作业优先级的作业时，该作

DOI:10.7544/issn1000-1239.2016.********
卷(期):起止页,年

业恢复执行。Liu 和 Layland 已经证明，EDF 调度算法的可调度利用率等于 1，EDF 调度算法也是一种最优的调度算法。EDF 调度算法在处理器利用率小于等于 1 的情况下能够实现最优调度。

然而，RM 算法和 EDF 算法都只适用于周期性任务调度且是单处理机的情况下。在实际应用中，需要考虑各种因素的影响，需要对上述算法进行扩展[13]。

2 周期性任务的调度架构

针对舆情信息获取等任务的周期性特征，本文提出一种面向云计算平台的周期性任务的调度框架 FPT，实现云计算平台下周期性任务的“一次调度、多次执行”。同时，还提出一种基于 RM-FFDU 的执行序列生成算法 RM-FFDU4PT，寻找满足批量周期性任务执行约束的最小资源集合并为集合中的资源生成任务执行序列。

2.1 FPT 调度架构

现有云计算架构未考虑周期性任务的时序特征。任务的周期性意味着可相对准确地预测出任务每次执行的时间点，使得“一次调度、多次执行”成为可能。本文提出的面向周期性任务调度框架（Framework for Periodic Tasks，简称“FPT”），如图 2 所示。该框架主要由任务调度器（Task Scheduler，简称“TS”）、资源管理器（Resource Monitor，简称“RM”）、模型解析器（Model Analyzer，简称“MA”）、执行序列生成器（Execution Sequence Generator，简称“ESG”）、本地管理器（Local Monitor，简称“LM”）等五个主要的功能模块组成。

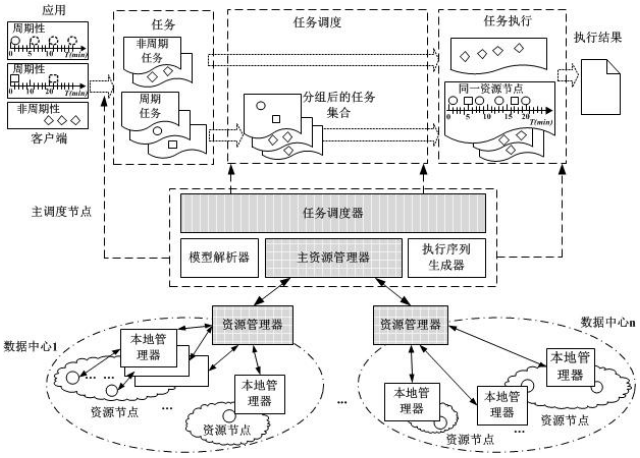

图 2. FPT 任务调度架构

Fig.2 FPT Task Scheduling Architecture

FPT 框架各模块功能描述如下：

1. 任务调度器（TS）。主要完成任务调度功能，对于非周期性任务调度过程不变，即 TS 根据一定的调度策略（如 FIFO、Max-Min 等算法）完成资源的分配。对于周期性任务则首先由模型解析器根据资源需求的不同进行分组，同一组的任务可在同一类型的资源上执行；然后，调用执行序列生成器模块以寻找满足执行要求的最小资源集合，并为集合中的每个资源生成周期性任务的执行序列；最后，将任务下发至相应云计算平台的资源上执行；

2. 资源管理器（RM）。主要负责收集数据中心各资源节点使用及各任务的执行情况，并作为任务调度的重要依据。为提高跨域云计算平台资源管理的效率，采用了两层管理架构，主资源管理器负责全局资源信息的管理及任务状态的监控，而各局点数据中心各有一个资源管理器，负责本局点资源信息的管理及任务状态的监控。资源管理器可以采用类似 Hadoop 集群中的心跳机制完成信息的收集，即由部署在各资源节点上的本地管理器以 5 秒钟或是 10 秒钟为间隔实时监测资源使用及任务执行信息并以心跳的形式发送给资源管理器。各资源管理器在收集到相关信息后也可以采用类似的心跳机制向主资源管理器发送信息。为避免因不同资源管理器在同一时刻向主资源管理器发送信息而产生拥塞现象，在各心跳周期内不同资源节点在发送心跳信息时可增加一个随机时延，以使信息“错峰发送”。为提高主资源管理器信息收集的能力，可增加“拉”的方式，即可按需向各资源管理器主动请求各数据中心内资源的状态信息。为提高资源管理的效率，Zhuo Zhang 等人[14]基于阿里飞天平台的已有架构设计了一种增量式的资源管理机制，可以大幅提升海量资源的管理效率；资源管理是云计算平台的核心研究内容之一，因本章主要探讨针对周期性任务的调度架构，资源管理并不是研究重点；

3. 模型解析器（MA）。该模块解析出任务对资源的需求，对于周期性任务需执行分组功能，将可在同一类型资源节点上执行的任务分成一组。解析的结果提供给任务调度器作为资源分配的依据。云计算平台中，资源的类别主要由操作系统的类型、所在的物理位置、所接入网络的基础运营商、网络接入方式、网络带宽等属性共同决定；

4. 执行虚列生成器（ESG）。在满足任务执行需求的前提下，为节约资源，同一组周期性任务需要用尽量少的资源节点来执行。然而，寻求满足上述要求的最小资源节点数是 NP 难的。文章[15]对此开展研究，将它看成是一个装箱问题，并用贪婪算法中的降序首次适应方法给出局部最优解。求得最小的资源集

DOI:10.7544/issn1000-1239.2016.********
卷(期):起止页,年

合之后，还需为集合中的每一个资源节点生成任务执行序列，以便于本地任务管理。ESG 是对周期性任务调度优化的重要模块，下一节将详细论述；

5. 本地管理器（LM）。LM 部署在跨域云计算平台的每个资源节点上并具体完成以下功能：定期采集节点资源使用情况，并以心跳的方式发送至资源管理器；接收主调度节点下发的任务，若需执行的是周期性任务，则还会同时收到一个执行序列；监控并定期以心跳的方式向资源管理器上报任务的执行情况，对于周期性任务则还需要根据执行序列定期启动任务实例，以保证各任务实例满足执行时限。

2.2 RM-FFDU4PT 调度策略

在 FPT 框架下，为资源节点生成任务执行序列是周期性任务优化的关键，本节在文献[15]的基础上提出一种名为 RM-FFDU4PT 的任务执行序列生成算法。

在满足任务执行约束条件下，用尽量少的资源完成批量任务的执行是众多任务调度算法的目标。那么，在跨域数据中心环境下，如何用尽量少的资源来执行批量周期性任务的执行，并满足各任务执行的截止时间要求呢？该问题是一个 NP 问题，即可以在多项式时间内判断一个解是否满足条件，但无法在多项式时间内求得该解。

其实，上述问题是一个装箱问题，即给定 n 种物品和 m 个箱子，每个箱子的负载为 M，第 i 个物品 i 的负载是 w_i，怎样才能使得能以最小数量的箱子数将 n 种物品装下？可把满足一组任务执行的资源看成箱子集合，每个周期性任务看成一个货物，而任务的处理机利用率看做是负载值，箱子装满即相当于在放入任何一个任务后该箱子中的任务会由可调度变为不可调度，如图 3 所示。

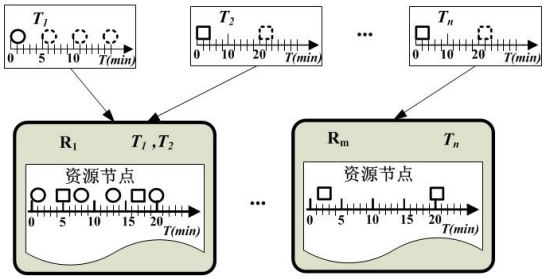

图 3. 周期性任务装箱过程

Fig.3 Periodic Task Scheduling Packing Process

对装箱问题求解方法的研究已有很多，本文按降序首次适应（First Fit Decreasing，FFD）算法求解，该算法的优势是简单且能高效地找到相对最优解。降序的参照值是周期性任务的处理机利用率，而判断箱子是否装满则以公式 1 为判断依据，其中 u_i 表示任务的处理器利用率。

$$u_n \le 2\left[\prod_{i=1}^{n-1}(1+u_i)\right]^{-1}-1$$

在找到最少资源集合后，每个资源内部可按照 RM 算法完成任务调度。RM 为一种静态优先级调度方法，周期大小越小任务优先级越高，会越优先调度执行。基于上述思路，本文提出一种面向周期性任务的降序首先适应算法（Rate Monotonic based First Fit Decreasing Utilization For Periodic Task，RM-FFDU4PT），能够以最少的资源数量完成任务分配，并为每个资源生成执行序列，如下表所示：

表 1 RM-FFDU4PT 算法伪代码

Table1 Pseudo-code Description for RM-FFDU4PT

算法 **RM-FFDU4PT**. 输入: 周期性任务集合 Γ（含 n 个任务）,资源集合 V; 输出: 资源个数 m, 链表- l_1 to l_m。

(1) Sort Γ in the order of non-increasing utilization//将 Γ 中任务按处理机利用率降序排列后得 $\Gamma=\{T_1, T_2, \cdots, T_n\}$
(2) i=1; m=1;
linkedlist l_1;
(3) j=1;
While(u_i>2($\prod_{r=1}^{k_j}(u_{j,r}+1)$)$^{-1}$-1) //任务 t_i 不能在 V_j 上执行
j=j+1;
(4) k_j= k_j +1; // k_j 是 V_j 上已分配需执行的任务数量，现需将任务分配至 V_j 执行，所以任务数量加 1
(5) if (j>m) //新增加了一个资源以执行任务，此时 m 值需要更新，并创建一个链表
{ m = j;
linkedlist l_j;}
l_j.addinOrder(t_i);//将任务 t_i 分配至 V_j 执行，l_j 有序保存各任务的下标值，对应任务周期越小则排序越靠前
u_{j,k_j} = u_i; //该值默认为 0，接纳任务 t_i 后需置为该任务的 u_i
(6) i=i+1;
(7) if (i>n) exit;//任务已分配完毕，退出
else goto (3). //为下一个任务分配资源

该算法生成执行序列的具体过程如下：

步骤 1.将任务集合 Γ 中的任务按时间利用率 u_i 降序排列；

步骤 2.为资源 j 创建链表 l_j；

DOI:10.7544/issn1000-1239.2016.********
卷(期):起止页,年

步骤 3.将任务集合 Γ 中的第一个任务分配至资源 1 上执行。在分配第二个任务开始，先判断公式 $u_i>2(\prod_{r=1}^{k_j}(u_{j,r}+1))^{-1}-1$ 是否成立。若成立表明该任务不能与目标资源上的任务一起执行，需要新增虚拟机执行该任务；否则将任务 t_i 分配至资源 j 上执行，并在链表 l_j 中按序加入任务编号；其中 u_j,r 表示分配给资源 j 的第 r 个任务的处理机利用率（r=1,2,…k_j, k_j 表示已分配至 j 的任务总数）；

步骤 4.循环执行步骤 3，直至任务分配完毕，各个资源链表中的任务序号即为执行序列。

RM-FFDU4PT 算法的时间复杂度为 O（nlogn），下一节将通过实验对该算法的性能进行评测。

3 实验分析

下面对 FPT 框架的效果和 RM-FFDU4PT 算法的有效性及其时间效率进行分析，所用负载为云计算平台承载的某个应用一个月的任务执行集。

3.1 FPT 架构测试

为评估 FPT 框架效果，本节重点对云计算平台所承载的某个应用 A 在一个月的任务下发情况进行分析，比较了在传统调度方式下和 FPT 框架下任务调度次数，结果如图 4 所示。应用 A 在一个月内共向云计算平台下发了 402.8 万个任务，日均近 13 万个，其中最多的一天近 16 万。在 FPT 框架下，共产生了 2.56 万次调度，日均调度 825 次，调度最多的一天发生在第一天，共产生 1.7 万次调度。应用 A 中的任务很多是需要重复、连续执行一个月、二个月甚至是更长的时间，实验负载截取了中间一个月的数据。因此，实验负载中绝大部分任务不是第一次执行。在 FPT 框架下调度时，仍然需要把这些任务当作是第一次执行，从而造成第一天的调度次数明显偏多。因此，为更为客观地对比优化效果，我们只考虑了在测试负载所在的一个月内应用新下发的任务，修正后的对比图如图 5 所示。

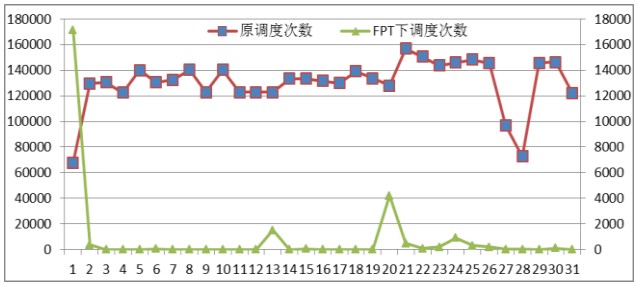

图 4. 优化前后任务调度次数对比

Fig.4 Original Task Scheduling Performance Before and After Optimization

图 5 中红色线表示原有调度次数（左侧为对应坐标轴），绿色线表示 FPT 下调度次数（右侧为对应坐标轴）。修正后，一个月内应用 A 新下发任务数为 61.2 万（都是周期性任务实例），在原调度框架下需调度 61.2 万次，而在 FPT 调度框架下只需调度 8300 次即可，即能将调度次数压缩至原调度次数的 1.36%。充分说明了 FPT 调度框架的优化效果。

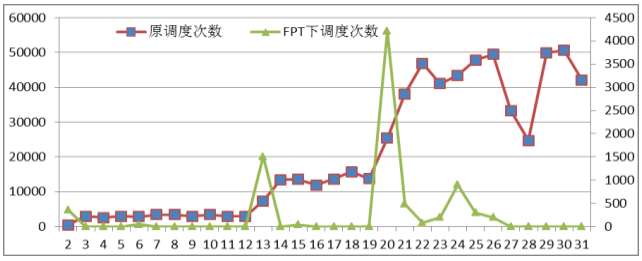

图 5. 修正后的任务调度次数对比

Fig.5 Revised Task Scheduling Performance Before and After Optimization

实际上，FPT 的优化效果取决于两个因素：总负载中周期性任务所占的比例、任务需持续运行时长与任务周期大小的比例($t_i.ft_i/t_i.pr_i$)。而对云计算平台中周期性任务分析得知，有近 50%的任务可看成是周期性任务的实例，任务的 ft_i 值即间断持续运行时长一般为二个月甚至是更长的时间，任务的 pr_i 值即周期大小一般为 1 个小时。若周期性任务持续运行时长为两个月，则传统调度框架下需调度任务的次数等于任务的执行次数即 24*60=1440，而在 FPT 框架下，理论上该周期性任务只需调度 1 次即可。

3.2 RM-FFDU4PT 算法测试

RM-FFDU4PT 是 FPT 框架的核心组成部分，本节着重对其运行效率进行分析。本文依托 jdk1.7.0_79 用 Java 语言实现了该算法，输入为同一组周期性任务集合，输出为所需资源个数及各个资源上运行的任务集合（按优先级大小排序）。评测的指标为算法调度不同数量的周期性任务所消耗的时间及占用的内存空间大小。

测试环境为一台曙光 A620r-F 服务器，配备双核 2.6GHz AMD2218CPU、4GB 内存，操作系统为 TurboLinux 3.4.3 版本。测试数据来源于云计算平台承载的某个应用一个月的任务负载，并按测试任务量量级的不同共开展了四轮测试。每轮测试分别随机选取 10 个、100 个、1000 个、10000 个任务各执行 10 次，并记载每次测试完成任务调度所需要的时间和空间大小，而后各求平均值作为该轮测试针对特定量级

DOI:10.7544/issn1000-1239.2016.********
卷(期):起止页,年

任务调度所需的时间及空间大小。统计结果如图 6 所示，横坐标为任务数量，主（左）纵坐标轴为内存占用大小，副（右）坐标轴为执行时间消耗。为便于比较，横纵坐标各取以 10 为底的对数刻度值。完成 10 个任务的调度所需时间不到 1 毫秒，占用内存 0.5MB；完成 1 万个任务的调度所需时间为 0.68 秒，占用内存为 5.36MB，在可承受范围之内。

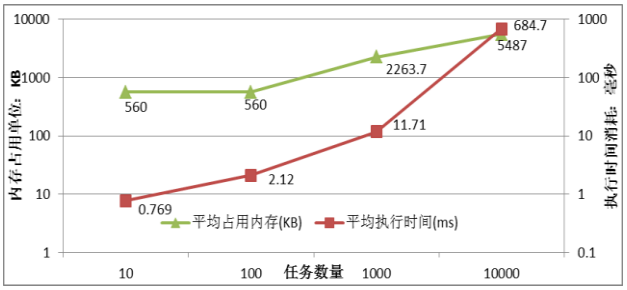

图 6. RM-FFDU4PT 算法效率统计

Fig.6 RM-FFDU4PT Execution Performance

然而，上述测试未考虑任务分组及在资源本地调度的时间消耗，主要原因有两点：首先，任务分组及在资源本地调度的时间都为 O(m)（m 为任务个数），小于 RM-FFDU4PT 的复杂度；其次，资源本地调度是由各本地管理器按照 ESG 生成的序列定时启动周期性任务来完成的，与现有嵌入式系统中 RM 算法的调度过程一样，资源消耗几乎可以忽略不计。

4 总结

在云计算环境下，资源和任务的异构性提升了任务调度的时间成本，加重了主调度节点负载，影响了系统吞吐率。在满足上层应用需求的前提下，尽量减少任务调度次数是降低调度负载的最根本、最直接的方法。然而，当前云计算平台所支持的绝大部分应用如科学计算、文本分析等任务的每一次执行都需要调度节点分配资源，无法减少调度次数，现有研究缺乏从降低调度次数入手对任务调度的优化。为此本文针对任务的周期性特征，提出了一种名为 FPT 的任务调度框架，设计了 RM-FFDU4PT 算法，能够以最少的资源满足批量周期性任务的执行需求，并为各资源生成执行序列，实现周期性任务的“一次调度、多次执行”，并通过实验说明了框架和算法的有效性。

参 考 文 献

[1] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[2] Tumanov A, Cipar J, Ganger G R, et al. alsched: Algebraic scheduling of mixed workloads in heterogeneous clouds[C]//Proceedings of the Third ACM Symposium on Cloud Computing. ACM, 2012: 25.

[3] Isard M, Prabhakaran V, Currey J, et al. Quincy: fair scheduling for distributed computing clusters[C]//Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. ACM, 2009: 261-276.

[4] Delimitrou C, Kozyrakis C. Paragon: QoS-aware scheduling for heterogeneous datacenters[J]. ACM SIGARCH Computer Architecture News, 2013, 41(1): 77-88.

[5] Delimitrou C, Sanchez D, Kozyrakis C. Tarcil: reconciling scheduling speed and quality in large shared clusters[C]//Proceedings of the Sixth ACM Symposium on Cloud Computing. ACM, 2015: 97-110.

[6] Schwarzkopf M, Konwinski A, Abd-El-Malek M, et al. Omega: flexible, scalable schedulers for large compute clusters[C]//Proceedings of the 8th ACM European Conference on Computer Systems. ACM, 2013: 351-364.

[7] Ousterhout K, Wendell P, Zaharia M, et al. Sparrow: distributed, low latency scheduling[C]//Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, 2013: 69-84.

[8] Hindman B, Konwinski A, Zaharia M, et al. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center[C]//NSDI. 2011, 11: 22-22.

[9] Vavilapalli V K, Murthy A C, Douglas C, et al. Apache hadoop yarn: Yet another resource negotiator[C]//Proceedings of the 4th annual Symposium on Cloud Computing. ACM, 2013: 5.

[10] Hung C C, Golubchik L, Yu M. Scheduling jobs across geo-distributed datacenters[C]//Proceedings of the Sixth ACM Symposium on Cloud Computing. ACM, 2015: 111-124.

[11] 谭朋柳. 开放式实时系统任务调度的研究[D]. 华中科技大学, 2008.

[12] Liu C L, Layland J W. Scheduling algorithms for multiprogramming in a hard-real-time environment[J]. Journal of the ACM (JACM), 1973, 20(1): 46-61.

[13] 邢建生, 刘军祥, 王永吉. RM 及其扩展可调度性判定算法性能分析[J]. 计算机研究与发展, 2005, 42(11): 2025-2032.

[14] Zhang Z, Li C, Tao Y, et al. Fuxi: a fault-tolerant resource management and job scheduling system at internet scale[J]. Proceedings of the VLDB Endowment, 2014, 7(13): 1393-1404.

[15] Oh Y, Son S H. Fixed-priority scheduling of periodic tasks on multiprocessor systems[J]. Department of Computer Science, University of Virginia, Tech. Rep. CS-95-16, 1995: 1-37.

作者介绍小五号

照片 Zhang peng, born in 1984. Associate Professor and Master Supervisor. His research interests include stream computing, distributed system and network security.

DOI:10.7544/issn1000-1239.2016.********
卷(期):起止页,年