文章编号：1674-2974（2023）08-0153-12 DOI：10.16339/j.cnki.hdxbzkb.2023280

基于MILP的应用服务器集群能耗与性能实时优化

熊智[1†]，赵敏[1]，蔡浩[1]，朱长盛[2]，许建龙[1]
（1. 汕头大学 工学院，广东 汕头 515063；
2. 汕头大学 科研处，广东 汕头 515063）

摘 要：在节能减排和激烈同行竞争的环境下，应用服务器集群的能耗与性能优化十分迫切.针对已有研究在性能指标和实时性方面的不足，提出一种集群能耗与性能实时优化方案.该方案结合采用线性加权法和主目标法优化集群功率与请求丢弃率这两个目标，将双目标优化转换成一个单目标约束优化.首先基于CPU频率等效连续调整模式下的服务器负载-功率模型，定义很少的变量将集群优化描述成混合整数二次规划问题，然后采用变量拆分和变量转换将其转化成混合整数线性规划（mixed integer linear programming，MILP）问题并引入特殊顺序集约束，最后采用Gurobi优化器求解该MILP.通过对CPU频率调整的进一步优化，大幅度减少了CPU频率的切换.多种场景下的测试表明，该方案的求解时间约在10 ms左右，特殊顺序集约束的引入使求解时间更为稳定，从而能够保证优化的实时进行.

关键词：应用服务器集群；能耗优化；约束优化；实时；混合整数线性规划；特殊顺序集约束

中图分类号：TP393 **文献标志码**：A

Real-time Optimization of Power and Performance for Application Server Clusters Based on MILP

XIONG Zhi[1†], ZHAO Min[1], CAI Hao[1], ZHU Changsheng[2], XU Jianlong[1]
（1. College of Engineering, Shantou University, Shantou 515063, China;
2. Scientific Research Management Division, Shantou University, Shantou 515063, China）

Abstract: In the environment of energy saving and fierce peer competition, it is very urgent to optimize the power and performance optimization of application server clusters. Aiming at the deficiencies of the existing research in performance indicators and real-time performance, a real-time optimization scheme of cluster power and performance was proposed. This scheme combined the linear weighting method and the master objective method to optimize the cluster power and request drop rate, so converting the bi-objective optimization into a single-objective constrainted optimization. Firstly, based on the server load-power model in the CPU frequency equivalent

* **收稿日期**：2022-09-26
基金项目：国家自然科学基金资助项目（61202366），National Natural Science Foundation of China（61202366）；广东省基础与应用基础研究基金资助项目（2021A1515012527），Guangdong Basic and Applied Basic Research Foundation（2021A1515012527）；广东省普通高校重点领域专项资助项目（2020ZDZX3073），Special Projects in Key Fields of Ordinary Colleges and Universities in Guangdong Province（2020ZDZX3073）
作者简介：熊智（1978—），男，湖北黄冈人，汕头大学教授，博士
† 通信联系人，E-mail：zxiong@stu.edu.cn

continuous adjustment mode, the cluster optimization was described as a mixed integer quadratic programming problem by defining few variables. Then, variable splitting and variable conversion were used to transform the problem into a MILP (mixed integer linear programming) problem, and we introduced an SOS (special-Ordered set) constraint. Finally, the Gurobi optimizer was used to solve the MILP problem. Through further optimization of CPU frequency adjustment, the switching of CPU frequency was greatly reduced. Tests in various scenarios showed that the average solution time of the scheme was approximately 10 ms and the introduction of SOS constraint made the solution time more stable, which can ensure the real-time optimization.

Key words: application server clusters; power optimization; constrained optimization; real-time; mixed integer linear programming; special-ordered set constraint

在Web系统中,处理动态请求的服务器称为应用服务器.动态请求对服务器的资源消耗较大,因而大型的Web系统,例如软件即服务(software as a service,SaaS)应用和大型网购平台等,通常采用应用服务器集群来提供服务.运营商通常按峰值容量部署应用服务器集群,但在实际运行中大多数时候服务器的利用率都很低,浪费了大量能耗.另一方面,应用服务器集群必须提供足够强大的性能以保证服务质量,否则会导致用户流失.在各个行业都节能减排[1-3]和同行竞争日益激烈的环境下,如何根据实际的负载状况实时优化应用服务器集群的部署,以在能耗与性能之间取得平衡是需要解决的重要问题.

CPU是服务器中能耗最大的部件[4],借助动态频率调整技术调低其工作频率可以降低其性能和运行功耗[5].另外,关闭多余的服务器也能有效减少集群功耗[6-7].有些研究仅依靠动态开关服务器[8-9],或仅依靠动态调整CPU频率[10-11]来优化集群的能耗与性能,还有些研究仅依靠任务调度[3]进行优化,它们都无法获得集群最优部署.在优化方法方面,常见的有如下三种:基于控制理论[12],基于阈值[8,13]和基于规划问题[7,10-11,14-16]的方法.在第一种方法中,由于服务器是一个非常复杂的非线性系统,并且需要同时控制性能和能耗,因而系统模型和控制方案都会非常复杂,需要测定大量的模型参数和控制参数,十分烦琐且可操作性差.至于第二种方法,通常基于排队长度、响应时间或利用率等阈值,采用启发式的方法调整服务器的开关和CPU频率,它们无法获得最优的部署.第三种方法将集群优化描述成规划问题,然后求解,该方法因能获得最优或者次优的部署而被广泛采用.

虽然在基于规划问题的集群优化方面已有一些较好的成果,但它们存在两方面的不足.第一方面是优化的性能指标.绝大多数相关研究都关注响应时间[8,11,14,16-18]而忽略了请求的丢弃,因为在无限容量的排队模型中,响应时间的公式较为简单,容易优化.然而在实际的Web系统中,Web请求的响应时间很短且服务器端通常采用多层结构,因而响应时间在很大程度上取决于用户的网络环境并且很难量化地进行优化.此外,实际的应用服务器当并发请求数达到预设的上限时会不再接受新的请求,因而并非是无限容量的.在这些研究中,文献[8]、[14]和[16]为请求的平均等待(或响应)时间设置上限,并将其作为约束条件,文献[11]以最小化请求的平均响应时间为优化目标.另外,文献[15]在给定功耗预算的前提下最大化评价函数,评价函数综合考虑了服务器的性能和效率.文献[3]考虑了服务器的CPU利用率,根据其值动态计算服务器的能耗效率.文献[17]虽然对集群能耗与请求丢弃率进行了综合优化,但存在下面第二方面的不足.

第二方面的不足在于优化的实时性.集群优化通常是一个复杂的NP-Hard问题,过长的求解时间会导致过高的性能违约率[19],并且当负载快速波动时需要及时调整集群部署,因此优化问题必须被快速求解.规划变量的定义和规划问题的求解是影响优化实时性的两个重要方面.已有的研究通常针对每个服务器定义变量[11,20],有的甚至为每个服务器的每个频率定义变量[7,21].在这些研究中,变量数目随着集群规模的变大而增加,从而导致大规模集群的优化问题无法实时求解.一般来说,集群中服务器型号的数量通常比服务器的数量低一到两个数量级,因为服务器通常是按型号一批一批地采购,如果能针对服务器的型号定义变量,变量数目将大幅度

减少.然而,有些研究仅考虑完全同构(即所有服务器同型号)的集群[10,22],它们的应用范围很有限.在规划问题的求解方面,有些研究仅采用启发式或贪婪算法求解[15,23-24],虽然求解速度快,但服务器参数和集群负载对求解质量影响很大.有些研究采用智能优化算法求解[25-27],然而其求解效率严重依赖算法参数的选择,并且无法保证能够得到全局最优解.还有些研究虽然提出了一些求解方法,但并未对求解效率进行测试[28].此外,有些研究关注高性能计算集群的优化[29-30],但Web请求与高性能计算任务不同,它们是短寿命的和大量的,因此资源无法被精确分配给每个Web请求.

针对上述不足,本文提出一种基于混合整数线性规划(mixed integer linear programming,MILP)的应用服务器集群能耗与性能实时优化方案,根据集群实际负载的大小动态调整各服务器的开关、CPU频率和负载.该方案结合采用线性加权法和主目标法来优化集群功率与请求丢弃率这两个目标,将双目标优化转换成一个带约束的单目标优化.我们首先基于服务器负载-功率曲线的特性得到两个结论,进而定义很少的变量将集群优化描述成一个混合整数二次规划(mixed integer quadratic programming,MIQP)问题,然后采用变量拆分和变量转换将其转化成MILP问题,最后采用Gurobi优化器[31]进行高效精确地求解.特殊顺序集(special-ordered set, SOS)约束的引入使得求解时间更加稳定,而CPU频率调整的进一步优化大幅度减少了CPU频率的切换.多种场景下的测试结果验证了所提出方案的可行性和有效性.

1　系统模型及分析

本文提出的优化方案采用基于规划问题的优化方法.在每个优化阶段结束时,优化控制器首先预测下个阶段的集群负载(即平均请求到达速率),然后将其带入规划问题进行求解,最后依据求解结果调整集群中各服务器的部署和负载分配方案.目前关于预测模型和Web负载预测的研究已有很多,这不在本文的讨论范围内.

1.1　集群能耗与性能优化模型

假设预测的集群负载为L.用S表示集群中所有服务器的集合,服务器s的负载和功率分别记为Load(s)和Power(s).考虑如下集群能耗与性能双目标优化:

$$\min_{\text{Deploymet of } s}\left[\sum\nolimits_{s\in S}\text{Power}(s),\ \frac{L-\sum_{s\in S}\text{Load}(s)}{L}\right], \tag{1}$$

$$\text{s.t.}\quad \sum\nolimits_{s\in S}\text{Load}(s)\leqslant L \tag{2}$$

第一个目标为集群的功率,第二个目标为集群的请求丢弃率,希望它们都尽可能小.

线性加权法和主目标法为多目标优化的两种常用方法,前者将多个子目标通过线性加权变成一个目标,后者选择最重要的一个子目标作为优化目标,并通过引入上界将其余子目标转化为约束条件.本文结合这两种方法,将集群能耗与性能优化描述如下:

$$\min_{\text{Deploymet of } s}\ \delta\sum_{s\in S}\text{Power}(s)+(1-\delta)\left[L-\sum_{s\in S}\text{Load}(s)\right], \tag{3}$$

$$\text{s.t.}\quad 0\leqslant\frac{L-\sum_{s\in S}\text{Load}(s)}{L}\leqslant U \tag{4}$$

目标函数(3)由两部分加权构成,前者为集群每秒的耗电量(即功率),后者为集群每秒的请求丢弃数,加权系数δ用于调节这两部分的重要程度.注意,这里将式(1)中的第二个目标由请求丢弃率改为了每秒的请求丢弃数,以使得两个目标描述的都是每秒的指标,从而使意义较为明确.并且,请求丢弃率和每秒的请求丢弃数之间只差一个比例常数(L是已知的),所以上述目标的修改相当于只是改变了权重的值,而权重本身是可调的.约束条件(4)的左边部分是约束条件(2)的变形,右边部分则是为第二个优化目标请求丢弃率引入的上界U.在实际应用中,过高的请求丢弃率会造成用户流失,因此为请求丢弃率设置一个上界十分必要且具有很好的实际意义.U和δ的值可根据实际应用场景进行设置.尤其是当U=0、δ=1时,上述优化就变成了性能保证下的能耗优化.优化方案的示意图如图1所示.

一旦依据求解结果部署好各服务器的开关和CPU频率,对于每个到达分配器的请求,它将以[L-$\sum$Load(s)]/L的概率被丢弃,以Load(s)/L的概率调度给服务器s.具体调度算法可以采用轮盘赌或者加权轮转来实现.

1.2　CPU频率调整模式和服务器负载-功率模型

现在的CPU通常都会提供一组电压和频率对的集合供用户选择,选择较低的电压和频率可降低CPU的性能和运行功率.另外,现在主流的CPU都是多核结构,为了便于能耗与性能的优化控制,和众多研究一样我们让其所有核的频率同时调整且保持相同.

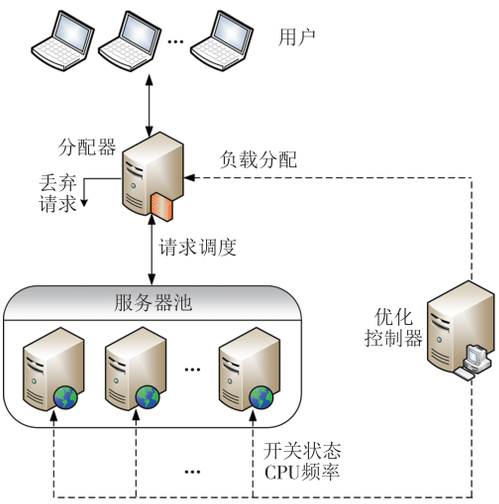


图 1 优化方案示意图

Fig.1 Schematic of the optimization scheme

CPU 可供选择的频率本文称为离散频率.当一台服务器工作在某个离散频率上,在请求丢弃率低到能忽略的情况下,它能承担的最大平均请求速率被定义为该频率的满载负载,承担满载负载时的功率称为该频率的满载功率.假设集群中的服务器有 M 个型号 $T_i(1\leqslant i\leqslant M)$,$T_i$ 型号服务器的相关参数如表 1 所示.

表 1 T_i 型号服务器的相关参数

Tab.1 The related parameters of servers of type T_i

符号	解释
N_i	服务器数量
C_i	CPU 离散频率的数量
$F_{i,j}$	从低到高的第 j 个离散频率,$1\leqslant j\leqslant C_i$
$L_{i,j}$	$F_{i,j}$ 的满载负载
$P_{i,j}$	$F_{i,j}$ 的满载功率
P_i^{spd}	挂起时的待机功率

对于一台 T_i 型号的服务器,假设该服务器分配到的负载为 l,$L_{i,j}<l<L_{i,j+1}$,为了保证不超载,应当让其 CPU 最低工作在离散频率 $F_{i,j+1}$ 上,但此时其能力过剩,无法最大限度地节能.为了更加细粒度地控制能耗与性能,本文采用如下方法让 CPU 频率等效连续调整从而让 CPU 满载工作:若一台 T_i 型号的服务器分配到的负载为 l,$L_{i,j}\leqslant l\leqslant L_{i,j+1}$,则让其 CPU 频率在 $F_{i,j}$ 和 $F_{i,j+1}$ 之间切换,且工作在 $F_{i,j}$ 和 $F_{i,j+1}$ 上的时间分别占 $(L_{i,j+1}-l)/(L_{i,j+1}-L_{i,j})$ 和 $(l-L_{i,j})/(L_{i,j+1}-L_{i,j})$.此时,该服务器的等效频率为

$$\mathrm{Frequency}_i(l)=\frac{L_{i,j+1}-l}{L_{i,j+1}-L_{i,j}}F_{i,j}+\frac{l-L_{i,j}}{L_{i,j+1}-L_{i,j}}F_{i,j+1} \tag{5}$$

功率为

$$\mathrm{Power}_i(l)=\frac{L_{i,j+1}-l}{L_{i,j+1}-L_{i,j}}P_{i,j}+\frac{l-L_{i,j}}{L_{i,j+1}-L_{i,j}}P_{i,j+1} \tag{6}$$

1.3 针对服务器型号定义变量的可行性

根据式(5)和式(6)可推导出服务器的频率-负载模型 $\mathrm{load}_i(f)$ 和频率-功率模型 $\mathrm{power}_i(f)$,前者为一系列点 $(F_{i,j},\ L_{i,j})$ 的分段线性插值,后者为一系列点 $(F_{i,j},\ P_{i,j})$ 的分段线性插值.根据这两组点的凸凹性可证明如下两个结论[27].

结论 1 对于所有承担负载在同一区间 $[L_{i,j},\ L_{i,j+1}]$ 的 T_i 型号服务器,让它们承担相同的负载可以得到集群能耗与性能优化的最优部署.

结论 2 对于开启的所有 T_i 型号服务器,让它们承担相同的负载可以得到集群能耗与性能优化的最优部署.

基于结论 1,可以针对每个服务器型号的每个负载区间定义变量.基于结论 2,可以针对每个服务器型号定义变量,进一步减少变量的数目.

2 基于 MIQP 的优化方案

基于结论 2,本节针对每个服务器型号定义变量,将集群能耗与性能优化描述成一个 MIQP 问题,然后设计求解方案.

2.1 优化问题的描述

针对 T_i 型号的服务器定义两个变量:整数 n_i 和实数 l_i,表示有 n_i 台该型号的服务器开启且以 l_i 的负载提供服务.引入变量 $p_i=\mathrm{Power}_i(l_i)$,那么优化模型(3)和(4)可描述成如下规划问题:

$$\min_{n_i,l_i,p_i}\delta\sum_{i=1}^{M}\left[n_ip_i+(N_i-n_i)P_i^{\mathrm{spd}}\right]+(1-\delta)\left[L-\sum_{i=1}^{M}(n_il_i)\right], \tag{7}$$

$$\text{s.t. } 0\leqslant\frac{L-\sum_{i=1}^{M}(n_il_i)}{L}\leqslant U \tag{8}$$

$$p_i=\mathrm{Power}_i(l_i),\ \forall i \tag{9}$$

$$n_i\in\{0,\cdots,N_i\},\ \forall i \tag{10}$$

$$l_i\in[L_{i,1},\ L_{i,C_i}],\ \forall i \tag{11}$$

式中:n_i 为整数变量,l_i 和 p_i 为实数变量,目标函数(7)和约束条件(8)均包含两个变量的相乘,故该问题是一个 MIQP 问题.该规划问题只需为每种服务器型号定义三个变量,变量数目非常少.注意到式(9)是一个函数型约束,虽然该约束本身比较复杂,但其引入使得目标函数(7)变得简单,使上述问题变成了一个标准形式的二次规划问题,便于采用已有的优化器进行求解.

2.2 基于Gurobi优化器的求解方法

很多相关研究讨论的是在保证性能的前提下最小化集群能耗,这类研究由于只有能耗一个优化目标,因而可以采用启发式策略得到较好的次优解,例如,优先开启能耗效率高的服务器并让其承担尽可能大的负载(以关闭尽可能多的能耗效率低的服务器),或者优先开启能耗效率高的服务器并让其工作在能耗效率尽可能高的频率和负载上(以取得尽可能高的能耗效率).然而,这里的优化目标(7)是集群能耗和性能的加权值,无法采用类似的启发式策略求解.

Gurobi是由Gurobi公司开发的新一代大规模数学规划优化器,支持MILP和MIQP等问题的精确求解,并提供C和Python等语言的API(application programming interface,应用程序接口).Gurobi支持多种类型的函数型约束,其中包括分段线性函数,因而能够描述约束(9).为了加快求解速度,我们采用Gurobi的C语言API实现了对规划问题(7)~(11)的求解.但该方法在一些场景下求解时间较长,无法满足集群实时优化的需求(参见4.2节中的测试).由于线性规划问题的求解开销较小,并且式(7)~(11)是一个二次问题且变量间存在分段线性的关系,因此我们希望能将其转化成线性规划问题进而求解.

3 基于MILP的优化方案

针对MIQP问题(7)~(11)的特点,本节通过变量拆分将分段线性函数变为线性函数,通过变量转换去掉变量的相乘,将该问题转化为一个MILP问题.接着,本节引入SOS约束,并对CPU频率调整做进一步的优化.

3.1 变量拆分

在式(7)~式(11)中,整数变量n_i和实数变量l_i表示有n_i台T_i型号的服务器在以l_i的负载提供服务.$\mathrm{Power}_i(l_i)$是一个分段线性函数,它在每个负载区间$[L_{i,j}, L_{i,j+1}]$上是线性函数,这种负载区间有C_i-1个,因此我们将n_i和l_i各拆分成C_i-1个变量$n_{i,j}$和$l_{i,j}$, $j=1, 2, \cdots, C_i-1$,表示有$n_{i,j}$台T_i型号的服务器在以$l_{i,j}$的负载提供服务,其中$l_{i,j}\in[L_{i,j}, L_{i,j+1}]$.这相当于仅使用结论1来针对每个服务器型号的每个负载区间定义变量.根据结论2可知,对于每个i,可让$n_{i,j}$中至多只有一个不为零,$j=1, 2, \cdots, C_i-1$.

记

$$\mathrm{Power}_{i,j}(l_{i,j})=\frac{L_{i,j+1}-l_{i,j}}{L_{i,j+1}-L_{i,j}}P_{i,j}+\frac{l_{i,j}-L_{i,j}}{L_{i,j+1}-L_{i,j}}P_{i,j+1} \tag{12}$$

则式(7)~式(11)可重写如下:

$$\begin{aligned}\min_{n_{i,j},l_{i,j}}\ &\delta\sum_{i=1}^{M}\sum_{j=1}^{C_i-1}(n_{i,j}p_{i,j})+\\&\delta\sum_{i=1}^{M}\left[(N_i-\sum_{j=1}^{C_i-1}n_{i,j})P_i^{\mathrm{spd}}\right]+\\&(1-\delta)\left[L-\sum_{i=1}^{M}\sum_{j=1}^{C_i-1}(n_{i,j}l_{i,j})\right],\end{aligned} \tag{13}$$

$$\text{s.t. } 0\leqslant\frac{L-\sum_{i=1}^{M}\sum_{j=1}^{C_i-1}(n_{i,j}l_{i,j})}{L}\leqslant U \tag{14}$$

$$p_{i,j}=\mathrm{Power}_{i,j}(l_{i,j}),\ \forall i,j \tag{15}$$

$$\sum_{j=1}^{C_i-1}n_{i,j}\leqslant N_i,\ \forall i \tag{16}$$

$$n_{i,j}\in\{0,\cdots,N_i\},\ \forall i,j \tag{17}$$

$$l_{i,j}\in[L_{i,j}, L_{i,j+1}],\ \forall i,j \tag{18}$$

3.2 变量转换

在优化问题(13)~(18)中,目标函数和约束条件中存在的二次项仅为$n_{i,j}p_{i,j}$和$n_{i,j}l_{i,j}$,由于$p_{i,j}$和$l_{i,j}$之间为线性关系,故$n_{i,j}p_{i,j}$和$n_{i,j}l_{i,j}$之间也是线性关系,我们希望通过合适的变量转换将$n_{i,j}p_{i,j}$和$n_{i,j}l_{i,j}$均变成线性函数.

根据1.2节的描述,$(L_{i,j+1}-l)/(L_{i,j+1}-L_{i,j})$和$(l-L_{i,j})/(L_{i,j+1}-L_{i,j})$分别表示服务器工作在频率$F_{i,j}$和$F_{i,j+1}$上的时间占比,我们将$(l-L_{i,j})/(L_{i,j+1}-L_{i,j})$称为切换因子,并记为$\eta$.显然切换因子$\eta\in[0,1]$,并且通过切换因子$\eta$能够计算出负载$l$.前面对$T_i$型号服务器的每个负载区间$[L_{i,j}, L_{i,j+1}]$定义两个变量$n_{i,j}$和$l_{i,j}$.我们用变量$\eta_{i,j}$替换掉变量$l_{i,j}$,它表示这$n_{i,j}$台服务器的总切换因子,也就是说这$n_{i,j}$台服务器的切换因子均为$\eta_{i,j}/n_{i,j}$,即

$$\frac{\eta_{i,j}}{n_{i,j}}=\frac{l_{i,j}-L_{i,j}}{L_{i,j+1}-L_{i,j}} \tag{19}$$

根据式(12)、式(15)和式(19)可进行如下推导:

$$\begin{aligned}&\textstyle\sum_{j=1}^{C_i-1}(n_{i,j}p_{i,j})+(N_i-\sum_{j=1}^{C_i-1}n_{i,j})P_i^{\mathrm{spd}}\\&=\textstyle\sum_{j=1}^{C_i-1}\left[n_{i,j}\left(\frac{L_{i,j+1}-l_{i,j}}{L_{i,j+1}-L_{i,j}}P_{i,j}+\frac{l_{i,j}-L_{i,j}}{L_{i,j+1}-L_{i,j}}P_{i,j+1}\right)\right]+\\&\quad(N_i-\textstyle\sum_{j=1}^{C_i-1}n_{i,j})P_i^{\mathrm{spd}}\\&=\textstyle\sum_{j=1}^{C_i-1}\left\{n_{i,j}\left[(1-\frac{\eta_{i,j}}{n_{i,j}})P_{i,j}+\frac{\eta_{i,j}}{n_{i,j}}P_{i,j+1}\right]\right\}+\\&\quad(N_i-\textstyle\sum_{j=1}^{C_i-1}n_{i,j})P_i^{\mathrm{spd}}\\&=\textstyle\sum_{j=1}^{C_i-1}\left[n_{i,j}(P_{i,j}-P_i^{\mathrm{spd}})+\eta_{i,j}(P_{i,j+1}-P_{i,j})\right]+\\&\quad N_iP_i^{\mathrm{spd}},\end{aligned} \tag{20}$$

$$n_{i,j}l_{i,j} = n_{i,j}L_{i,j} + \eta_{i,j}(L_{i,j+1} - L_{i,j}) \tag{21}$$

则式(13)~式(18)变为:

$$\begin{aligned}\min_{n_{i,j},\,l_{i,j}}\ & \delta\sum_{i=1}^{M}\sum_{j=1}^{C_i-1}\left[n_{i,j}(P_{i,j} - P_i^{\mathrm{spd}})\right] + \\ & \delta\sum_{i=1}^{M}\sum_{j=1}^{C_i-1}\left[\eta_{i,j}(P_{i,j+1} - P_{i,j})\right] + \\ & \delta\sum_{i=1}^{M}\left(N_iP_i^{\mathrm{spd}}\right) + (1-\delta)L - \\ & (1-\delta)\sum_{i=1}^{M}\sum_{j=1}^{C_i-1}\left(n_{i,j}L_{i,j}\right) - \\ & (1-\delta)\sum_{i=1}^{M}\sum_{j=1}^{C_i-1}\left[\eta_{i,j}(L_{i,j+1} - L_{i,j})\right],\end{aligned} \tag{22}$$

$$\text{s.t.}\quad 0 \leqslant \frac{L - \sum_{i=1}^{M}\sum_{j=1}^{C_i-1}\left[n_{i,j}L_{i,j} + \eta_{i,j}(L_{i,j+1} - L_{i,j})\right]}{L} \leqslant U \tag{23}$$

$$\sum_{j=1}^{C_i-1} n_{i,j} \leqslant N_i,\ \forall i \tag{24}$$

$$\eta_{i,j} \leqslant n_{i,j},\ \forall i,j \tag{25}$$

$$n_{i,j} \in \{0,\cdots,N_i\},\ \forall i,j \tag{26}$$

$$\eta_{i,j} \geqslant 0,\ \forall i,j \tag{27}$$

目标函数(22)是线性的,约束条件(23)~(25)也都是线性的,因此上述模型是一个MILP问题.我们同样采用Gurobi优化器对该MILP问题进行求解.并非所有的二次规划都能转化成线性规划,这取决于具体问题的形式.这里给出的转换方案对变量间存在线性约束的规划问题提供了一种参考求解方案.

3.3 SOS约束

虽然求解MILP问题(22)~(27)能得到集群的最优部署,但该问题的描述没有使用结论2.在3.1节中已分析到,如果使用结论2,那么可以增加如下约束:对于每个i, $n_{i,j}$中至多只有一个不为零,j=1, 2, …, C_i-1.SOS约束是一种高度专业化的约束,对给定有序集合里的变量可以接受的值进行限制.SOS约束有两种类型,其中SOS1型约束用于规定一个变量集合中最多只允许一个变量取非零值.Gurobi优化器提供GRBaddsos函数给规划问题增加SOS约束,因此我们采用该函数为变量$n_{i,j}$增加SOS1型约束.虽然增加该SOS约束与否并不会影响集群优化的结果,但其在一些场景下可以提高规划问题的求解效率.

注意到,$L_{i,j}$同时属于区间$[L_{i,j-1},\ L_{i,j}]$和$[L_{i,j},\ L_{i,j+1}]$.假设在优化结果中有10台T_0型号的服务器开启,它们的工作频率为$F_{0,2}$,负载为$L_{0,2}$.如果不采用SOS约束,求得的解可能为:$n_{0,1}$=4,$\eta_{0,1}$=4,$n_{0,2}$=6,$\eta_{0,2}$=0,其他$n_{0,j}$(j≠1和2)均为0.如果采用了SOS约束,则不可能出现上述情形,只会要么$n_{0,1}$=10,$\eta_{0,1}$=10,其他$n_{0,j}$(j≠1)均为0,要么$n_{0,2}$=10,$\eta_{0,2}$=0,其他$n_{0,j}$(j≠2)均为0.

3.4 CPU频率调整的进一步优化

上述方案对T_i型号服务器的每个负载区间$[L_{i,j},\ L_{i,j+1}]$定义了两个变量$n_{i,j}$和$\eta_{i,j}$,表示开启的$n_{i,j}$台服务器的CPU频率均在$F_{i,j}$和$F_{i,j+1}$之间切换,且时间占比分别为$1-\eta_{i,j}/n_{i,j}$和$\eta_{i,j}/n_{i,j}$.虽然CPU频率的切换可以通过调用CPUFreqUtils工具包提供的函数实现,但会有一定的开销,并且过于频繁地切换CPU频率有可能影响CPU寿命.根据式(19)可知,这$n_{i,j}$台服务器的总负载为

$$\begin{aligned}n_{i,j}l_{i,j} &= n_{i,j}L_{i,j} + \eta_{i,j}(L_{i,j+1} - L_{i,j}) \\ &= (n_{i,j} - \eta_{i,j})L_{i,j} + \eta_{i,j}L_{i,j+1}\end{aligned} \tag{28}$$

根据式(12)、式(15)和式(19)可知,这$n_{i,j}$台服务器的总功率为

$$\begin{aligned}n_{i,j}p_{i,j} &= n_{i,j}\left[(1 - \tfrac{\eta_{i,j}}{n_{i,j}})P_{i,j} + \tfrac{\eta_{i,j}}{n_{i,j}}P_{i,j+1}\right] \\ &= (n_{i,j} - \eta_{i,j})P_{i,j} + \eta_{i,j}P_{i,j+1}\end{aligned} \tag{29}$$

注意到$\eta_{i,j}$≤$n_{i,j}$,下面根据$\eta_{i,j}$的优化结果分两种情况对这$n_{i,j}$台服务器的部署进行调整.

情况1:$\eta_{i,j}$为整数.将$\eta_{i,j}$台的负载调整为$L_{i,j+1}$,CPU频率调整为$F_{i,j+1}$;将$n_{i,j}-\eta_{i,j}$台的负载调整为$L_{i,j}$,CPU频率调整为$F_{i,j}$.

情况2:$\eta_{i,j}$不为整数.将$\lfloor\eta_{i,j}\rfloor$($\lfloor*\rfloor$表示向下取整)台的负载调整为$L_{i,j+1}$,CPU频率调整为$F_{i,j+1}$;将$n_{i,j}-\lfloor\eta_{i,j}\rfloor-1$台的负载调整为$L_{i,j}$,CPU频率调整为$F_{i,j}$;将剩下1台的负载调整为$[1-(\eta_{i,j}-\lfloor\eta_{i,j}\rfloor)]L_{i,j}+(\eta_{i,j}-\lfloor\eta_{i,j}\rfloor)L_{i,j+1}$,CPU频率在$F_{i,j}$和$F_{i,j+1}$之间切换,且时间占比分别为$[1-(\eta_{i,j}-\lfloor\eta_{i,j}\rfloor)]$和$\eta_{i,j}-\lfloor\eta_{i,j}\rfloor$.

容易验证,经上述部署调整后,这$n_{i,j}$台服务器的总负载和总功率均保持不变,但至多只有一台服务器的CPU频率需要在$F_{i,j}$和$F_{i,j+1}$之间切换,大幅度减少了CPU频率的切换.进一步地,在上述情况2中,可以直接让最后的那一台服务器也工作在频率$F_{i,j+1}$上,虽然这样会多消耗一点点的能耗,但换来的是(在一个优化周期内)无需切换CPU频率.

4 测试

本节首先分别测试基于MIQP和基于MILP的优化方案的求解时间,尤其展示将MIQP转换成MILP和引入SOS约束这两个措施的效果,最后将提出的方案与已有的基于MILP的方案进行对比测试.

4.1 测试环境

表2列出了6种服务器的频率、性能和功率数据[7,27],各服务器的待机功率约为3.5 W.我们使用3

个不同规模的集群进行测试,它们均由这6种型号的服务器构成,各型号的数量相等,分别为40、160和640台,即这3个集群的规模分别为240、960和3 840台服务器.请求丢弃率的上限U设置为10%.优化控制器为Intel Core I7-6700 CPU和32G内存的PC机,安装Windows 10系统,Gurobi的版本为9.1.2.

表2　各型号服务器的频率、性能和功率数据

Tab.2　Frequency, performance, and power data for each server type

型号	CPU	频率/GHz	满载负载/(req·s^{-1})	满载功率/W
1	AMD Athlon 64 3 500+	1.0, 1.8, 2.0, 2.2	51.2, 91.2, 101.4, 111.4	74.7, 95.7, 103.1, 110.6
2	AMD Athlon 64 3 800+	1.0, 1.8, 2.0, 2.2, 2.4	53.8, 95.4, 104.8, 113.6, 122.3	75.2, 89.0, 94.5, 100.9, 107.7
3	AMD Athlon 64 5 000+	1.0, 1.8, 2.0, 2.2, 2.4, 2.6	99.4, 177.4, 197.2, 218.0, 234.6, 255.2	82.5, 99.2, 107.3, 116.6, 127.2, 140.1
4	Pentium-M 1.8G	0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8	37.3, 50.0, 62.4, 74.4, 88.4, 97.6, 111.4	44.0, 45.0, 47.0, 49.0, 51.0, 55.0, 60.0
5	Intel Core I5-3450	1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1	190.3, 202.0, 212.4, 224.8, 235.9, 247.7, 258.1, 269.1, 290.5, 301.6, 309.2, 318.7, 326.3, 337.9, 344.4	47.8, 48.7, 49.6, 50.5, 51.5, 52.5, 53.6, 54.8, 57.4, 58.9, 60.6, 62.4, 64.3, 66.4, 68.7
6	Intel Xeon E3-1230	0.8, 1.0, 1.2, 1.3, 1.5, 1.7, 1.9, 2.0, 2.2, 2.4, 2.6, 2.8, 2.9, 3.1, 3.3	159.2, 195.6, 229.8, 244.2, 275.6, 301.7, 324.0, 337.4, 358.9, 379.4, 398.5, 417.8, 422.2, 436.7, 448.1	57.9, 59.8, 61.9, 63.0, 64.9, 67.3, 70.2, 71.2, 73.6, 77.3, 80.6, 83.8, 85.2, 88.4, 92.0

4.2　基于MIQP优化方案的求解时间

对于一个集群来说,各个服务器最高频率的满载负载之和为该集群能承担的最大负载,称为集群最大负载.下文用集群实际负载与集群最大负载的比值来表示负载的大小,例如,50%的集群负载是指集群负载等于集群最大负载的50%.

集群规模,集群负载,以及加权系数δ都对规划问题的求解时间有影响.在每个集群规模下,我们用20%、50%和80%三种负载测试基于MIQP优化方案的求解时间;在每种负载情况下,又让加权系数δ以0.01的间隔从0变到1.每个场景下的测试重复10次,求解时间取平均值,测试结果见图2.

从图中可以看到,大致上求解时间随着集群规模的增大而变长,这主要是因为规划变量n_i的范围变大了,而求解时间与集群负载和加权系数之间没有明显规律.绝大多数场景下的求解时间超过了50 ms,有些场景下的求解时间甚至接近4 s,因此基于MIQP的优化方案无法保证优化的实时性.

4.3　基于MILP优化方案的求解时间

接下来测试基于MILP优化方案的求解时间.为了分别展示将MIQP转换成MILP和引入SOS约束这两个措施的效果,本小节先不采用SOS约束,而在下个小节专门测试引入SOS约束的效果.与4.2节一样,在每个集群规模和集群负载下,我们让加权系数δ以0.01的间隔从0变到1分别进行测试,然后将求解时间的平均值、中值、最小值和最大值与基于MIQP的方案进行对比,测试结果见表3.表3中的结果表明,无论是求解时间的平均值和中值,还是最小值和最大值,基于MILP的优化方案均明显低于基于MIQP的优化方案,尤其是平均值和中值降到了10 ms左右.然而对比最大值和中值来看,基于MILP的优化方案的求解时间不太稳定,在某些场景下最大值为中值的11倍多.

4.4　引入SOS约束的效果

本小节的测试场景(包括集群规模、集群负载和加权系数)与前两个小节一样.对于每个场景,分别测试不采用SOS约束和采用SOS约束这两种情况下基于MILP优化方案的求解时间,结果见图3.

在图3的子图(a)、(e)、(g)、(h)和(i)中,采用SOS约束后平均求解时间变长了.这是因为在这5幅子图中不采用SOS约束时的求解时间已经很小(平均值不超过8.1 ms),SOS约束的引入使得模型本身变得更加复杂,从而导致求解时间略有增加.在其他4幅子图中,采用SOS约束均使得平均求解时间有所缩短,尤其在子图(c)和(f)中,平均求解时间从14.4 ms和12.9 ms分别缩短为9.7 ms和8.1 ms.

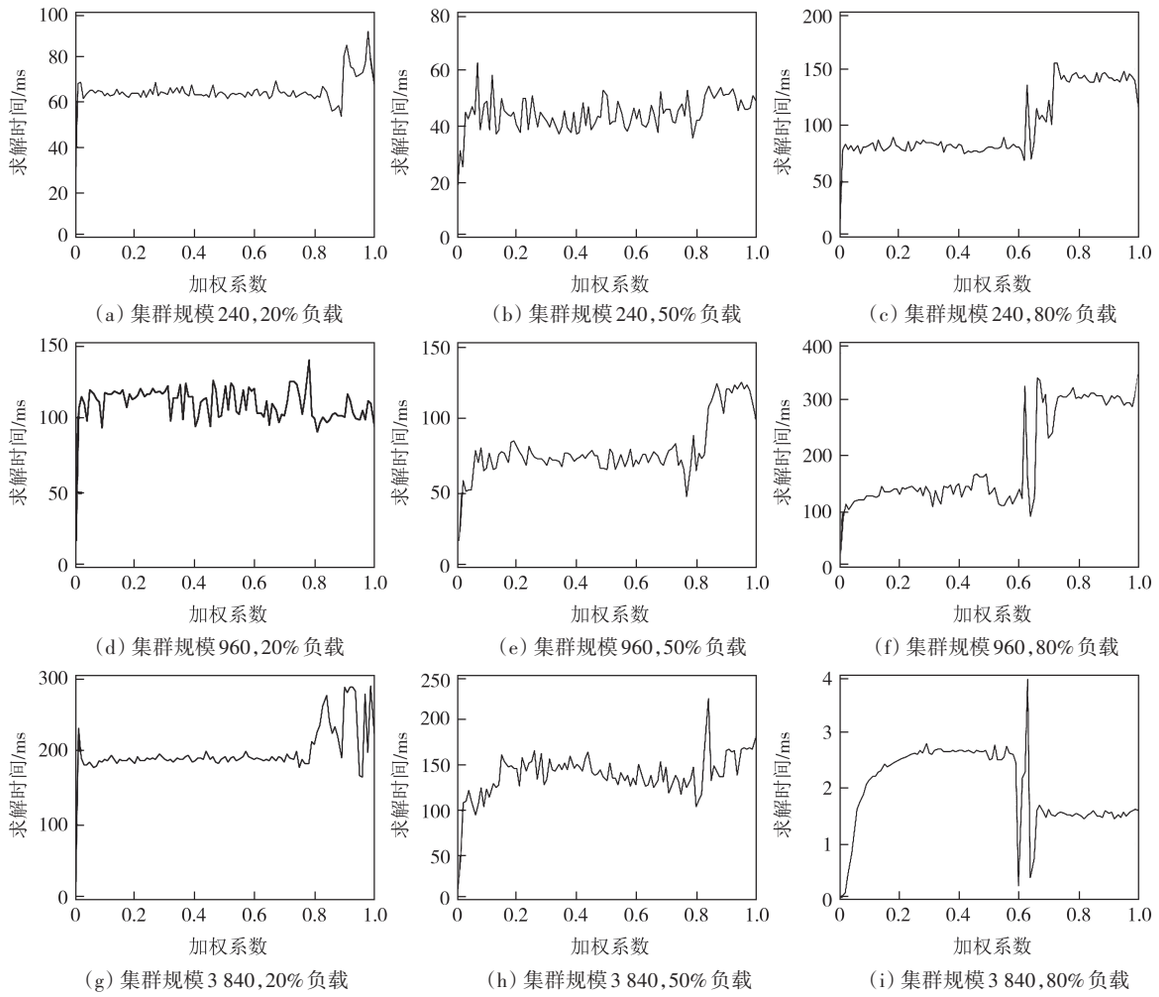


图2 MIQP方案的求解时间

Fig.2 Solution time for the MIQP scheme

表3 变为线性规划后求解时间的降低

Tab.3 Reduction of solution time after changing to linear programming

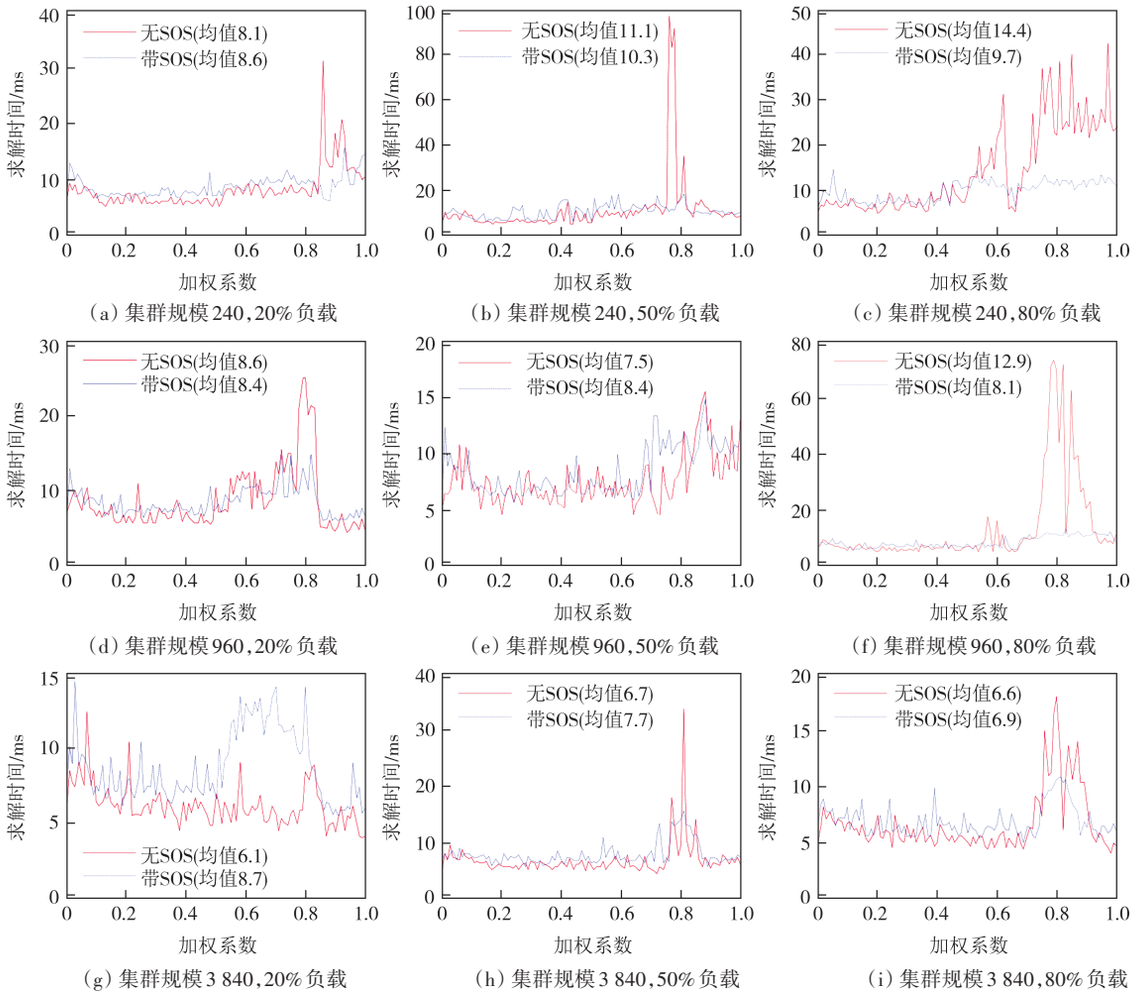
集群规模	集群负载/%	MIQP方案的求解时间/ms				MILP方案(无SOS约束)的求解时间/ms			
		平均值	中值	最小值	最大值	平均值	中值	最小值	最大值
240	20	64.5	63.8	19.0	91.0	8.1	7.3	4.9	31.1
	50	43.9	44.0	16.1	62.3	11.1	8.5	4.5	97.3
	80	98.8	82.3	12.5	154.1	14.4	9.5	4.8	42.4
960	20	108.5	111.5	15.3	138.4	8.6	7.0	4.3	24.9
	50	78.3	74.4	12.6	124.3	7.5	6.9	4.4	15.5
	80	189.4	141.5	12.6	353.3	12.9	6.8	4.5	73.4
3 840	20	197.0	188.3	18.3	286.6	6.1	5.9	4.0	12.4
	50	138.0	138.3	10.4	223.6	6.7	6.0	4.3	33.8
	80	1 975.1	2 171.5	19.4	3 907.4	6.6	5.6	4.0	18.0

当不采用SOS约束时,求解时间在一些场景下会突然变得很长,而当引入了SOS约束后,求解时间则比较稳定.该现象在子图(a)、(b)、(c)、(d)、(f)、(h)和(i)中非常明显.

综合来看,由于不采用SOS约束时的求解时间已经很短,故引入SOS约束对平均求解时间没有明显的改善,但其使得求解时间变得稳定,能更好地保证求解的实时性.

4.5 与其他基于MILP方案的对比

文献[7]和[21]针对每个服务器的每个频率或



(a) 集群规模240,20% 负载　(b) 集群规模240,50% 负载　(c) 集群规模240,80% 负载

(d) 集群规模960,20% 负载　(e) 集群规模960,50% 负载　(f) 集群规模960,80% 负载

(g) 集群规模3 840,20% 负载　(h) 集群规模3 840,50% 负载　(i) 集群规模3 840,80% 负载

图3　SOS约束的效果

Fig.3　The effect of the SOS constraint

每对相邻频率区间定义3个变量,其中2个实数变量,1个二进制变量,进而将集群能耗与性能优化描述成MILP问题.我们采用该变量定义方式(即针对服务器定义变量)将模型(3)~(4)描述成一个MILP问题,并采用Gurobi优化器进行求解.我们对该方案进行了测试,测试场景和方法与4.3节、4.4节一样.由于过长的求解时间没有实用价值,因此求解时间的上限设置为1 min.该方案与本文基于MILP的方案在变量数目上的对比见表4,在求解时间上的对比见表5.在表5中,超时场景的求解时间不参与平均值、中值、最小值和最大值的计算,超时率等于超时场景数除以每个集群规模和负载下的总场景数101.

在前面的测试中,请求丢弃率的上限U被固定为10%.为了更加全面地测评本文方案的求解实时性,我们对U取5%和15%的情形也分别进行了测试,结果见表6和表7.

在表4中,由于其他基于MILP的方案针对服务器定义变量,因而当集群规模变大时,变量的数目随之增加.而本文基于MILP的方案针对服务器型号定义变量,只要集群中服务器的型号没发生变化,变量的数目就不会发生变化.因为本文基于MILP方案的变量数目少,因此在表5~表7中,无论是求解时间的平均值和中值,还是最小值和最大值,本文的方案均优于其他基于MILP的方案.尤其当应用于后两个规模较大的集群时,其他基于MILP的方案均出现了一些求解时间超时的情形.此外,从表5~表7中还可以看到,当U取5%和15%时,本文方案的求解时间及

表4　变量数目对比

Tab.4　Comparison of the number of variables

集群规模	其他基于MILP方案的变量数目		本文基于MILP方案的变量数目	
	二进制变量	实数变量	整数变量	实数变量
240	2 080	4 160	46	46
960	8 320	16 640	46	46
3 840	33 280	66 560	46	46

表5 求解时间对比(U=10%)

Tab.5 Comparison of solution times (U=10%)

集群规模	集群负载/%	超时率/%	其他基于MILP方案的求解时间/ms				本文基于MILP方案的求解时间/ms			
			平均值	中值	最小值	最大值	平均值	中值	最小值	最大值
240	20	0.0	774.6	74.3	58.1	3 044.5	8.6	8.4	5.9	15.5
	50	0.0	990.6	82.0	64.6	2 599.1	10.3	10.1	5.8	18.0
	80	0.0	1 180.0	1 270.8	62.1	3 721.4	9.7	10.1	6.0	14.6
960	20	4.0	11 091.8	275.6	233.3	56 165.4	8.4	7.9	5.9	14.8
	50	19.8	2 556.1	336.5	283.5	55 049.5	8.4	7.5	5.6	14.8
	80	14.9	9 741.6	268.3	254.6	59 692.9	8.1	7.4	5.5	13.0
3 840	20	5.9	6 439.1	7 012.8	1 129.9	9 261.8	8.7	7.9	5.4	14.4
	50	4.0	1 641.4	1 489.4	1 137.9	5 631.3	7.7	7.1	5.4	15.5
	80	5.9	6 777.5	2 480.0	1 144.0	47 673.4	6.9	6.4	5.0	10.8

表6 求解时间对比(U=5%)

Tab.6 Comparison of solution times (U=5%)

集群规模	集群负载/%	超时率/%	其他基于MILP方案的求解时间/ms				本文基于MILP方案的求解时间/ms			
			平均值	中值	最小值	最大值	平均值	中值	最小值	最大值
240	20	0.0	519.9	70.8	54.5	1 414.9	8.2	7.9	5.6	12.9
	50	0.0	1 674.0	80.8	66.9	7 775.3	10.8	11.6	5.8	17.7
	80	0.0	1 463.0	1 284.3	59.9	5 075.0	9.3	9.9	5.8	13.4
960	20	15.8	14 304.8	286.8	246.3	56 665.9	8.9	8.7	6.7	13.7
	50	8.9	1 863.8	360.3	302.9	49 117.9	8.0	7.0	5.6	12.7
	80	0.0	4 235.0	295.6	270.7	52 387.6	6.8	6.3	5.3	10.7
3 840	20	15.8	7 068.9	7 340.5	1 083.1	8 811.1	9.6	8.9	6.4	14.9
	50	6.9	2 210.4	1 721.7	1 236.0	25 058.1	7.7	6.7	5.1	17.6
	80	1.0	3 382.4	2 541.9	1 260.5	6 878.7	6.1	5.9	5.1	11.0

表7 求解时间对比(U=15%)

Tab.7 Comparison of solution times (U=15%)

集群规模	集群负载/%	超时率/%	其他基于MILP方案的求解时间/ms				本文基于MILP方案的求解时间/ms			
			平均值	中值	最小值	最大值	平均值	中值	最小值	最大值
240	20	0.0	886.1	70.5	54.3	5 200.8	9.0	8.3	6.2	17.0
	50	0.0	932.2	88.2	66.8	2 741.9	10.5	11.1	5.7	17.0
	80	0.0	547.6	195.2	59.2	3 575.5	8.6	8.1	5.4	14.5
960	20	18.8	15 889.5	243.4	231.0	55 960.3	8.9	8.8	6.3	14.2
	50	9.9	2 726.8	333.4	253.9	59 754.8	8.1	7.0	5.7	12.9
	80	9.9	11 615.7	286.1	268.6	58 918.2	7.9	6.8	5.0	11.9
3 840	20	12.9	8 014.1	7 574.2	1 083.3	49 688.5	9.6	9.0	6.4	15.3
	50	4.0	2 031.7	1 519.5	1 225.5	7 677.5	7.8	6.8	5.7	14.5
	80	5.0	6 589.8	7 963.4	1 306.4	9 216.6	6.7	6.1	5.1	12.4

其波动程度均与U取10%时的相当,因此U值的设置不会对本文方案的优化实时性产生影响.

5 结 论

本文首先结合线性加权法和主目标法设计了一种应用服务器集群能耗与性能优化模型,然后基于服务器负载-功率曲线的特性得到了两个结论,进而提出了两种优化方案:基于MIQP的优化方案与基于MILP的优化方案,它们均采用Gurobi优化器对规划问题进行精确求解.前者只须针对每种服务器型号定义3个变量,因而变量数目很少,但由于是非线性问题,当应用于集群规模很大的一些场景时求解时间过长.后者针对MIQP问题的特点,采用变量拆分

和变量转换将其转化成一个MILP问题,并引入了SOS约束.测试结果表明,将MIQP转换为MILP后,平均求解时间降到了10 ms左右,而SOS约束的引入使得求解时间更加稳定,能够保证规划问题的实时求解.与已有的基于MILP的优化方案相比,本文提出的方案大幅度减少了变量数目并缩短了求解时间.此外,本文虽然让CPU在相邻离散频率之间来回切换以等效工作在连续频率上,并在该模式下建立服务器功率模型和集群优化模型,但在最终部署时对CPU的频率调整做了进一步优化,大幅度减少了CPU频率的切换.接下来,我们将研究如何根据应用场景的不同选择合适的加权系数.

参考文献

[1] 周舟,袁余俊明,李方敏. 云数据中心服务器能耗建模及量化计算[J]. 湖南大学学报(自然科学版),2021,48(4):36-44.
ZHOU Z, YUAN Y J M, LI F M. Energy consumption modeling and quantitative calculation of servers in cloud data center[J]. Journal of Hunan University (Natural Sciences),2021,48(4):36-44.(in Chinese)

[2] JIN C Q, BAI X L, YANG C, et al. A review of power consumption models of servers in data centers[J]. Applied Energy,2020,265:114806.

[3] LIN W W, WANG W Q, WU W T, et al. A heuristic task scheduling algorithm based on server power efficiency model in cloud environments[J]. Sustainable Computing: Informatics and Systems,2018,20:56-65.

[4] VASQUES T L, MOURA P, DE ALMEIDA A. A review on energy efficiency and demand response with focus on small and medium data centers[J]. Energy Efficiency,2019,12(5):1399-1428.

[5] 王静莲,龚斌,刘弘,等. 软硬件节能原理深度融合之绿色异构调度算法[J]. 软件学报,2021,32(12):3768-3781.
WANG J L, GONG B, LIU H, et al. Green heterogeneous scheduling algorithm through deep integration of hardware and software energy saving principles[J]. Journal of Software,2021,32(12):3768-3781.(in Chinese)

[6] O'DWYER K J, CREEDON E, PURCELL M, et al. Power saving proxies for web servers[J]. The Computer Journal,2020,63(2):179-192.

[7] BERTINI L, LEITE J C B, MOSSÉ D. Power optimization for dynamic configuration in heterogeneous web server clusters[J]. Journal of Systems and Software,2010,83(4):585-598.

[8] 孙健,廖丹,李可,等. 基于排队论的异构数据中心性能及能源管理策略[J]. 电子科技大学学报,2018,47(2):161-168.
SUN J, LIAO D, LI K, et al. A strategy for queuing theory-based performance and energy management in heterogeneous data centers[J]. Journal of University of Electronic Science and Technology of China,2018,47(2):161-168.(in Chinese)

[9] CHANG C J, CHANG F M, KE J C. Optimal power consumption analysis of a load-dependent server activation policy for a data service center[J]. Computers & Industrial Engineering, 2019, 130:745-756.

[10] LI K Q. Power and performance management for parallel computations in clouds and data centers[J]. Journal of Computer and System Sciences,2016,82(2):174-190.

[11] CAO J W, LI K Q, STOJMENOVIC I. Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across clouds and data centers[J]. IEEE Transactions on Computers,2014,63(1):45-58.

[12] SHI X Y, DONG J, DJOUADI S M, et al. PAPMSC: power-aware performance management approach for virtualized web servers via stochastic control[J]. Journal of Grid Computing, 2016, 14(1): 171-191.

[13] RAJAGOPAL T K P, VENKATESAN M. Energy efficient server with dynamic load balancing mechanism for cloud computing environment[J]. Wireless Personal Communications, 2022, 122(4):3127-3136.

[14] ZHOU Z, LIU F M, ZOU R L, et al. Carbon-aware online control of geo-distributed cloud services[J]. IEEE Transactions on Parallel and Distributed Systems,2016,27(9):2506-2519.

[15] CIESIELCZYK T, CABRERA A, OLEKSIAK A, et al. An approach to reduce energy consumption and performance losses on heterogeneous servers using power capping[J]. Journal of Scheduling,2021,24(5):489-505.

[16] GU C L, LI Z L, HUANG H J, et al. Energy efficient scheduling of servers with multi-sleep modes for cloud data center[J]. IEEE Transactions on Cloud Computing,2018,8(3): 833 - 846.

[17] 熊智,赵悦源,许建龙,等. 应用服务器集群能耗与性能平衡的在线实时优化[J]. 控制与决策,2021,36(11):2589-2598.
XIONG Z, ZHAO Y Y, XU J L, et al. Online real-time optimization of power-performance tradeoff for application server clusters[J]. Control and Decision, 2021, 36(11): 2589-2598.(in Chinese)

[18] ENTRIALGO J, MEDRANO R, GARCÍA D F, et al. Autonomic power management with self-healing in server clusters under QoS constraints[J]. Computing,2016,98(9):871-894.

[19] MONTEIRO A, LOQUES O. Quantum virtual machine: power and performance management in virtualized web servers clusters[J]. Cluster Computing,2019,22(1):205-221.

[20] WANG P J, QI Y, LIU X. Power-aware optimization for heterogeneous multi-tier clusters[J]. Journal of Parallel and Distributed Computing,2014,74(1):2005-2015.

[21] XIONG Z, CHENG Y, CAI L R, et al. Online power-aware deployment and load distribution optimization for application server clusters[J]. IEEE Access,2019,7:91080-91092.

[22] WANG W, ABDOLRASHIDI A, YU N P, et al. Frequency regulation service provision in data center with computational flexibility[J]. Applied Energy,2019,251:113304.

[23] HAO Y S, CAO J, MA T H, et al. Adaptive energy-aware scheduling method in a meteorological cloud [J]. Future Generation Computer Systems,2019,101:1142-1157.

[24] DONG Z Q, LIU N, ROJAS-CESSA R. Greedy scheduling of tasks with time constraints for energy-efficient cloud-computing data centers[J]. Journal of Cloud Computing,2015,4(1):1-14.

[25] QI L Y, CHEN Y, YUAN Y, et al. A QoS-aware virtual machine scheduling method for energy conservation in cloud-based cyber-physical systems[J]. World Wide Web, 2020, 23(2): 1275-1297.

[26] 胡志刚,肖慧,李克勤. 云计算中基于多目标优化的虚拟机整合算法[J]. 湖南大学学报(自然科学版),2020,47(2):116-124.
HU Z G, XIAO H, LI K Q. Virtual machine consolidation algorithm based on multi-objective optimization in cloud computing[J]. Journal of Hunan University (Natural Sciences), 2020,47(2):116-124.(in Chinese)

[27] XIONG Z, ZHAO M, TAN L, et al. Real-time power optimization for application server clusters based on Mixed-Integer Programming[J]. Future Generation Computer Systems, 2022, 137: 260-273.

[28] TIAN Y, LIN C, LI K Q. Managing performance and power consumption tradeoff for multiple heterogeneous servers in cloud computing[J]. Cluster Computing,2014,17(3):943-955.

[29] HAO M, ZHANG W Z, WANG Y M, et al. Fine-grained powercap allocation for power-constrained systems based on multi-objective machine learning [J]. IEEE Transactions on Parallel and Distributed Systems,2021,32(7):1789-1801.

[30] AZIMI R, JING C, REDA S. PowerCoord: power capping coordination for multi-CPU/GPU servers using reinforcement learning[J]. Sustainable Computing: Informatics and Systems, 2020,28:100412.

[31] Gurobi Optimization, LLC. The fastest solver — Gurobi[EB/OL]. [2023-05-06]. https://www.gurobi.com.