面向突发业务的云服务并发量应对策略研究

郭 军[1] 武 静[1] 邢留冬[2] 张 斌[1] 张 荣[1]

[1](东北大学计算机科学与工程学院 沈阳 110819)

[2](麻省大学电子与计算机工程学院 达特茅斯 美国 02747-2300)

摘 要 云服务突发并发量是导致服务质量(QoS)降级的重要因素.传统的突发并发量应对策略存在时效性差、资源利用率低的问题,会导致服务请求响应时间增长、请求违例率及拒绝率增大.针对上述问题,该文首先提出了基于多级队列的并发量分级缓存机制 ATBM,可有效避免突发并发量对缓存队列的持续拥堵,确保缓存队列中请求的持续流通,进而提高整体的 QoS.在此基础上,提出了一个主动式突发并发量应对策略 GMAC,该策略能以较低的欠预测比例及规模预测用户并发量,得出欠分配规模较低的资源需求量,并根据系统当前的资源配置、并发量处理情况以及 SLA(Service Level Agreement),全面、准确地量化资源可用量,最后通过对比资源需求量及可用量,提前制定及执行资源调整策略.该文采用真实并发量数据对 ATBM 在改善 QoS 方面的有效性进行验证,最后利用对比实验就 GMAC 的突发并发量应对效果进行评估.实验结果表明,ATBM 下并发量的违例率仅为传统单级缓存队列下违例率的 24.6%;相对于根据并发量峰值预留,GMAC 可在保证服务不发生 SLA 违例的前提下,将系统资源利用率提高 1.45 倍;与基于并发量均值预留资源相比,GMAC 能将系统的平均响应时间及拒绝率分别降低 8.6% 及 16.9%.

关键词 云服务;突发并发量;QoS;资源调整;欠预测

中图法分类号 TP301 **DOI 号** 10.11897/SP.J.1016.2019.00864

A Coping Strategy for Bursty Workload of Cloud Service

GUO Jun[1] WU Jing[1] XING Liu-Dong[2] ZHANG Bin[1] ZHANG Rong[1]

[1](*School of Computer Science and Engineering, Northeastern University, Shenyang* 110819)

[2](*Department of Electrical and Computer Engineering, Massachusetts University, Dartmouth*, 02747-2300, *USA*)

Abstract Bursty workload is a crucial factor that deteriorates QoS (Quality of Service) of cloud services. However, traditional coping strategies for bursty workloads suffer disadvantages as poor timeliness and low resource utilization rate, which not only lengthens request response time but also increases request violation and rejection rate. To solve these problems, we first propose a hierarchical VM cache mechanism named as ATBM, and then give a proactive resource adjustment strategy named as GMAC, both of which improve QoS under cloud services when they are caught in bursty workloads. What's more, instead of using a single queue to cache requests within VMs, ATBM divides cache queues into two kinds: buffer queues and block queues. The former is set to cache requests, and the latter is applied to block requests. This multi-level cache mechanism efficiently avoids bursty workload's long-term congestions to cache queues, and ensures continuous circulation of requests as cached in queues. And on the foundation of ATBM, GMAC is designed which consists of two significant components: a prediction algorithm called as MGM for resources

收稿日期:2017-05-29;在线出版日期:2017-12-01.本课题得到国家自然科学基金(61300019,61370155)、中央高校东北大学基本科研专项基金(N120804001)资助.**郭 军**,男,1974 年生,博士,副教授,中国计算机学会(CCF)会员,主要研究方向为云计算、服务计算.E-mail:guojun@mail.neu.edu.cn.**武 静**(通信作者),女,1992 年生,硕士,主要研究方向为云计算、突发业务处理及优化.E-mail:neu_15_wujing@126.com.**邢留冬**,女,1975 年生,博士,教授,博士生导师,主要研究领域为云计算、可靠性建模及复杂系统分析.**张 斌**,男,1964 年生,博士,教授,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为云计算、服务计算.**张 荣**,女,1992 年生,硕士,主要研究领域为云计算、服务预测和优化.

that are demanded to keep QoS, and a maximum available resource evaluation model named as ACM. Furthermore, MGM is based on a modified Grey Model which greatly inherits the high-robust lying in GM (Grey Model) that means having no strict limitation to the types of workloads. At the same time, MGM makes full use of residuals of GM to improve prediction results. Consequently, MGM is able to predict workload with low under-prediction rate and scale, which makes its prediction in resource also hold a small under-estimate scale and rate. On the other hand, ACM conclude its analysis in available resources by considering initial resource configurations of VMs, current requests processing conditions and SLA (Service Level Agreement) comprehensively as well as reasonably. As a result, GMAC depends on MGM to get the number of resources that demanded to ensure QoS, and utilize ACM to evaluate the available number of resource within current systems. Then, by means of comparing the quantities of demanded resources and available resources to get the number of resources calling for being adjusted, GAMC plans and carries out the resource adjustment strategy in advance through adding or removing VMs. Finally, we employ real workload data to detect the effectiveness of ATBM to enhance system QoS in the aspects of decreasing requests violation together with rejection rate, and optimizing the structure of request response that makes more requests to be responded within shorter time. And we also conduct comparison experiments to analyze the validity of GMAC in coping bursty workload. The results show that the violation rate of ATBM is only 24.6% of that from traditional single cache queue. Meanwhile, compared with resource reservation strategies based on workload peaks, GAMC increases resource utilization rate by 1.45 times without any SLA violations. And the average response time and rejection rate of the cloud service decrease by 8.6% and 16.9% with no violations, when it switches its resource adjustment strategy from reserving resources relying on the means of workloads to GMAC.

Keywords　cloud service; bursty workload; quality of service; coping strategy; under-estimate prediction

1　引　言

云计算众多的优良特性，如大规模的资源共享、按需分配的服务方式等，使得越来越多的服务由本地迁至云端[1-2]. 云服务数量的增大、类型的增多，使得其对应的并发量变化规律也日渐多样、复杂.

规律性较低、单位时间内瞬时激增的并发量，我们称之为突发并发量，突发并发量广泛存在于各类云服务[3]. 维基百科关于迈克尔·杰克逊相关内容的访问量在其逝世当天及随后一天由先前的不足 17 000(次/天)分别增加到 1 447 904(次/天)及 5 875 404(次/天)①. 突发并发量会导致请求响应时间增长、请求违例率②及拒绝率③增加；在多层级应用中，突发并发量更会导致层级之间资源利用率失衡，即部分资源过载而部分资源空闲，进而造成系统 QoS(Quality of Service)降级. 例如 Shopping mix 场景下的模拟用户达到 110 时，Front Server 的利用率到达 100%，而后端 Database Server 的利用率却仅有 40%[4].

弹性是云服务的基本属性，它使得云服务可根据并发量的实时变化，动态、弹性地调整资源分配，来保证系统的 QoS，同时实现资源利用率的最大化[5-7]. 与云服务相比，传统的集群服务并未实现资源的按需分配和弹性调整. 本文以突发并发量场景下，云服务与传统的集群服务的突发并发量对应策略为例，阐述两类服务模式的区别. 集群服务通常会基于并发量峰值对应的资源需求量峰值，确定部署规模. 在整个服务生命周期中，无论并发量如何“动态波动”，资源供给量始终处于“静止”的状态. 这种资源状态和数量的不匹配、不对等，在低并发时因

① http://stats.grok.se/en/200906/Michael%20Jackson
② 请求违例率指请求响应时间超过 SLA 规定值的用户并发量数占总用户并发量数的比例.
③ 请求违例率指由于 VM 无可用缓存空间而被拒绝的用户并发量数占总用户并发量数的比例.

资源利用率过低而造成资源浪费、能耗过大的问题，在高并发时因资源不足而导致 QoS 降级. 然而，云服务却能有效解决上述问题，它会根据并发量的实时变化，动态使用或释放资源，即当突发并发量到达时，及时追加资源，突发并发量过后，再释放资源，以保证系统的 QoS，同时合理使用资源.

具体而言，云服务提供商通常会采用如下两种突发并发量应对策略：一种是被动式的资源伸缩[5,8]，该方法要求系统预先设定一系列的触发条件及调整动作，当且仅当触发条件满足时，调整动作才会被执行. 由于云环境状态的不断变化，该策略的应对效果不可避免地存在一定程度的“滞后性”[1,6,9]；而这种“滞后性”在大规模突发并发量中，体现得尤为明显. 这是因为突发并发量规模的增大会使得系统瞬时启动的资源量增多，进而造成总体的资源启动及初始化时间增长. 该段时间内系统资源量将处于“供小于求”的状态，系统的 QoS 也便无法保证.

另一种是主动式的资源预留，其又可细分为基于统计模型的资源预留[10-13]及基于预测算法的资源预留[6-7,14]. 其中，前者通常根据并发量峰值确定资源预留量，但当并发量峰值和均值差距较大且突发并发量发生频率较低时，大量的预留资源将被长期闲置，系统的资源利用率也便随之降低. 后者则首先预测并发量，然后基于预测结果提前制定、执行相应的资源调整动作. 资源预留的“前瞻性”使得资源在实际并发量到达之前便可开始部署，从而降低了资源初始化对 QoS 的影响，改善了调整策略的时效性. 与此同时，与统计模型下静态的、基于并发量峰值的资源预留相比，预测式资源预留针对实时并发量进行动态地、针对性地调整，有效地避免了资源的长期闲置，提高了系统资源利用率.

预测式资源预留策略在确定资源调整量时，需要完成两项工作：预测资源需求量和评估资源可用量. 一方面，资源需求量预测通常要先对系统的并发量进行预测，然后根据并发量的类型及规模，得出相应的资源需求量. 该过程中常用的并发量预测算法有：ARMA[15-16]、ARIMA[17-19,9]、GM(Grey Model)[20-22]、混合 GM(Hybrid GM)[23]、神经网络(neural network)[24-25]、深度学习[26]、遗传算法[27]、SVR(Support Vector Regression)[28-29]等. 然而，这些预测算法存在以下不足：(1) 对并发量数据的规律性要求过强；例如，ARMA、ARIMA 要求数据是平稳的或经过差分处理后可以转变为平稳，然而实际的并发量很难满足这些条件；(2) 预测算法的欠预测比例过高、规模过大，即并发量预测值严重低于实际值；预测算法的预测原理是对数据潜在规律的挖掘及延续，如利用逻辑回归拟合并发量的变化规律，然后利用该规律进行预测. 突发并发量的不可预知性及瞬时激增性会使预测算法不可避免地出现欠预测. 其中，并发量预测算法欠预测比例及规模的增大，会导致资源需求量预测结果欠预测比例及规模的增加. 与此同时，系统会取资源需求量和资源可用量的差值作为最终的资源调整量. 当资源需求量处在较高的欠预测水平时，即使系统能保证高精度的资源可用值评估结果，其仍将执行欠评估的资源调整策略，导致系统资源量处于“供小于求”的状态，引发系统 QoS 降级. 因此，为了保证资源调整策略的有效性，系统需要增强预测算法的鲁棒性，同时降低并发量预测算法的欠预测比例及规模.

另一方面，系统需要对资源可用量进行评估. 目前，大部分的评估算法仅分析当前的资源分配量，如当前服务占用的 VM 数或 VM 创建时的原始配置，并默认该资源分配量为下一时刻的资源可用量. 然而，资源分配量和资源可用量并不一定相等，且通常是前者大于后者. 其原因在于，系统当前正在处理的请求在该时钟周期可能无法完成，它们需要顺延到下一时钟周期继续进行处理. 这些被顺延的请求会占用部分资源，导致其无法用于下一周期最新到达的请求的处理. 故而，使得下一周期的资源可用量小于资源分配量. 因此，为了更准确地评估资源可用量，我们需要充分考虑请求处理过程的顺延情况. 一方面，我们需要保证被顺延的请求不发生 SLA 违例；另一方面，我们希望系统接入并处理尽可能多的新到达的请求，以提高系统的资源利用率.

针对上述问题，本文提出了一个基于多级队列的并发量分级缓存机制 ATBM. ATBM 使用多级队列对并发量进行分级缓存，并为不用的层级赋以不同的处理优先级，通过对 VM 中缓存并发量的合理调度及处理，有效地避免了突发并发量对缓存队列的持续拥堵，进而保证了 VM 的 QoS.

在此基础上，本文提出了一个预测式的资源预留策略 GMAC. GMAC 首先采用基于改进的灰度预测算法 MGM 对资源需求量实施预测，然后基于

最大可接受并发量评估模型 ACM 对资源可用量进行评估，最后通过对比资源需求量及可用量，制定并执行相应的调整策略. 其中，MGM 在继承灰度模型 GM 对预测数据类型及变化规律高鲁棒性的基础上，充分挖掘预测残差中的欠预测信息，以降低 GM 欠预测的比例及规模. ACM 则综合分析 VM 当前的缓存状况及资源配置，求出足以保证 SLA（请求拒绝率及违例率）的最大可接受并发量，进而得出系统的资源可用量.

本文第 2 节将介绍突发并发量应对策略的相关工作；第 3 节给出 ATBM 的工作原理，并在第 4 节介绍 ACM 的计算方法；第 5 节给出 GMAC 的实现细节；第 6 节通过对比 ATBM 及传统单级队列下请求的平均响应时间、违例率、拒绝率及资源利用率，验证前者在改善 QoS 方面的有效性；与此同时，该节会对 ACM 计算结果的准确性、GMAC 在提高资源利用率及保证突发并发量下 QoS 的有效性进行验证；第 7 节将对全文进行总结，并介绍下一步的研究内容.

2　相关工作

2.1　被动式的资源伸缩

被动式的资源伸缩由于实现过程简单，其在工业界得到广泛应用，如 Amazon Auto Scaling①. 学术界则更多地关注于提高资源伸缩规则的合理性及改善调整效果的实时性. Hasan 等人[8]通过实现阈值的自适应调整来避免 VM 频繁调整；Grechanik 等人[30]通过构建并发量类型及数量与资源需求类型及数量之间的关系模型，进行资源弹性调整. Yang 等人[31]提出了一个基于应用特点且可有效避免资源竞争的资源伸缩框架. Sun 等人[32]以及 Coutinho 等人[33]构建了包含 CPU 及 RAM 利用率、请求违例率的触发条件，以及相应的触发阈值和对应的调整动作；当触发条件满足时，系统会将当前的 VM 数输入斐波那契数列，通过该数列求得需要额外创建的 VM 数. 然而，被动式资源伸缩策略存在时效性差等方面的不足. 该类调整策略通常包括两个方面内容：触发条件及相应的调整动作；当且仅当满足触发条件时，系统设置的各项资源调整动作才会被执行. 但是，从并发量规模或系统某些属性（如 CPU 利用率、RAM 利用率）满足触发条件，资源调整策略被执行，到最终产生实际效果，需要消耗大量时间. 该时段内系统的资源供给量处于“供小于求”的状态，此时的 QoS 也便无法保证. 因此，本文主要采用时效性更高的主动式资源预留作为资源调整手段，即提前预测并发量，并在并发量到达之前制定、执行相应的资源调整策略，以保证系统的服务效果.

2.2　主动式的资源预留

时间序列中的预测模型 ARMA[15-16]、ARIMA[9,17-19]、GM（Grey Model）[20-22]、混合 GM（Hybrid GM）[23]. Roy 等人[16]使用 2 阶的 ARMA 对并发量进行预测. Calheiros 等人[9]中使用 R 提供的 ARIMA 的接口对并发量进行预测. 灰度理论也可进行时间序列分析，在过去的几十年间其被广泛应用于各个领域，其中包括金融、农业、医疗、军事等. Wei 等人[20]使用 GM 及卡尔曼过滤对无线传感器中 sensor node 和 sink node 的数据量进行预测. Tseng 等人[23]中使用 GM(1,1)及移动平均比去季节法（ratio-to-moving-average-deseasonalization method）预测具有季节因子的时间序列. 然而，这些预测算法在突发并发量下都会出现严重的欠预测，造成资源欠分配. 另一方面，众多机器学习算法、智能算法，如神经网络（neural network）[24-25]、深度学习[26]、遗传算法[27]、SVR（Support Vector Regression）[28-29]、随机森林（Random Forest）[34]、梯度提升决策树（GBDT）[35]等也被用于并发量预测. 这些算法虽然能保证较高的预测精度，但是它们预测过程的时间开销过大，资源调整策略的时效性过低. 为此，本文提出了一个改进的灰度预测算法 MGM，来预测系统资源需求量. MGM 在继承灰度模型 GM 对预测数据类型、变化规律高鲁棒性及预测过程高时效性的基础上，充分挖掘预测残差中的欠预测信息，以降低算法的欠预测比例及规模.

大量的统计模型，如 3 次样条插值[36]、多项式分析[37]、线性回归[38]、MMPP（2）[10,39-42]、SVM（Support Vector Machine）[43]等也被应用于资源预留. Ali-Eldin 等人[36]中使用三次样条函数（cubic spline）拟合以周为重复模式的并发量（repetitive weekly pattern），并为预测残差构建自回归模型 AR（autoregressive model），最后用样条函数及 AR 之和代表预测值. Sladescu 等人[37]分析了“拍卖

① Amazon Elastic Computer Cloud. http://aws.amazon.com/ec2/

活动”以及“足球赛事”期间访问量的变化，并使用多项式模型对每个阶段的并发量进行建模. Huang 等人[38]中使用线性回归对云平台下的资源需求量进行预测. Zhang 等人[10]将 VM 在突发并发量下的资源需求量和非突发状态下的资源需求量构建为 MMPP(2)模型，然后基于该模型构造马尔可夫过程，用以描述系统处于突发并发量下 VM 数量及对应的概率，最后根据上述过程确定足以保证系统 QoS 的资源预留量. 但是，如果 VM 并发量的峰值和均值差距较大且突发并发量发生频率较低时，大量的预留资源将被长期闲置，系统的资源利用率便会大幅降低. 为此，本文提出了一个根据并发量规模实施资源动态、弹性调整的资源预留策略 GMAC. GMAC 一方面利用其核心部件 MGM 对资源需求量进行预测；另一方面会调用内部的 ACM 模型对资源可用量进行评估，通过对比资源需求量和可用量，确定最终的资源分配量.

3 基于多级队列的缓存机制 ATBM

突发并发量具有不可预知性和瞬时激增性，仅存在单一缓存队列的传统 VM 缓存机制在面临突发并发量时，会出现缓存队列持续拥堵，导致后续非突发并发量无法进行处理、只能排队等待的问题，进而引发请求响应时间增长、拒绝率增大，甚至是 SLA 违例的问题. 因此，本文提出一个基于多级队列的并发量分级缓存机制 ATBM. ATBM 使用多级队列对并发量进行分级缓存，并为不用的层级赋以不同的处理优先级，通过对 VM 中缓存并发量的合理调度及处理，有效地避免了突发并发量对缓存队列的持续拥堵，提高了 VM 应对突发并发量的能力.

3.1 ATBM 的组成要素

ATBM 对 VM 的缓存机制了改进，将 VM 传统的单一缓存(SQ)转化为包含缓存及阻塞的多级缓存. 如图 1 所示，ATBM 由多级队列组成，底层为缓存队列(cache)，上层为阻塞队列($block_i$，$i=1,2,3,4$).

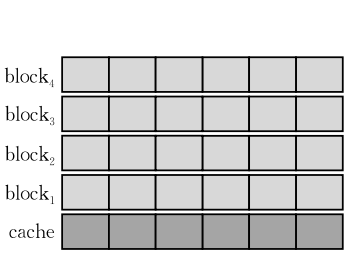


图 1 ATBM 的组成结构

当请求进入 ATBM 时，会优先搜索底层的缓存队列 cache；当 cache 中无可用空间时，才会逐层搜索阻塞队列，直至找到可用空间以完成存储，或搜索失败而被 VM 拒绝，即整个 ATBM 被充满已无可用空间. 其中，缓存队列中的请求具有更高的处理优先级，阻塞队列中的请求处理优先级较低，且层级越高，对应的优先级越低. 与此同时，当且仅当底层缓存队列有空闲空间时，存储于阻塞队列中的请求才会逐层滑入缓存队列，进行处理.

单级模式 SQ 的存储队列不区分缓存队列和阻塞队列，请求到达 SQ 后会查看 VM 是否有可用的存储空间：若有空间，完成存储；若无可用空间，该请求被拒绝.

本文使用序列 $P=\{\langle t_1,C_1\rangle,\langle t_2,C_2\rangle,\langle t_3,C_3\rangle\}$ 代表并发量访问模式，$\langle t_i,C_i\rangle$ 表示 t_i 时刻有 $C_i\times q$ 个请求访问系统，其中 q 为队列长度.

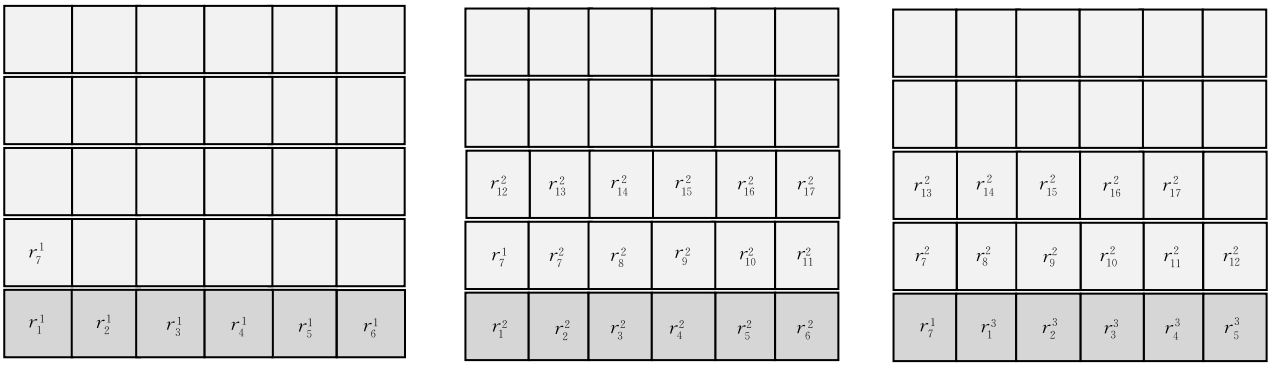
基于 ATBM 和 SQ 的缓存及阻塞原理，可计算两者在不同访问模式下请求的总延迟，并基于总延迟计算平均延迟，求解方法如式(1)所示：

$$d_{avg}=(d_1+d_2+d_3)/(C_1+C_2+C_3)/q \quad (1)$$

其中 d_1,d_2,d_3 分别表示并发量 qC_1,qC_2,qC_3 访问系统时对应的总延迟，q 为队列长度，d_{avg} 为求得的平均延迟.

假设 ATBM 各层队列长度 q 均为 6，即每层队列均可存储 6 个请求，VM 单位时间内可处理 6 个请求；队列级数为 5，即有 1 层缓存队列、4 层阻塞队列. 其中，并发量访问模式 $P=\{\langle t_1,C_1=1.1\rangle,\langle t_2,C_2=2.9\rangle,\langle t_3,C_3=0.9\rangle\}$，即 t_1、t_2、t_3 时刻分别有 $\{r_i^1|i=1,\cdots,7\}\{r_j^2|j=1,\cdots,17\}\{r_k^3|k=1,\cdots,5\}$ 个请求访问系统，存储过程如图 2 所示. t_1 时刻到达的请求依次存入 cache 及 $block_1$，存于 cache 中的请求不存在延迟；(t_1,t_2) 时段内，cache 中的请求处理完成并返回. t_2 时刻涌入 17 个请求，其充满 cache 的同时，占用了 $block_1$ 及 $block_2$，ATBM 中出现小规模拥堵；(t_2,t_3) 时段内，cache 中的请求处理完成. t_3 时刻到达的请求数较少，cache 出现空余空间；上层阻塞队列 $block_1$ 及 $block_2$ 中的请求便依次向下滑动，部分存入 cache，并在该时钟周期完成处理.

本文以请求 r_1^1 及 r_7^1 的处理过程为例，给出 ATBM 和 SQ 下请求延迟的计算方法. 在 ATBM 中，r_1^1 于 t_1 时刻到达后便可进行处理，故其请求延迟为 0；r_7^1 在 t_1 时刻到达，t_3 时刻才能完成处理，故对应的请求延迟为 2. 如图 3 所示，在 SQ 中，r_1^1 于 t_1 时刻



(a) t_1时刻ATBM的存储状态　(b) t_2时刻ATBM的存储状态　(c) t_3时刻ATBM的存储状态

图 2　ATBM 的存储及请求处理过程

(a) t_1时刻SQ的存储状态

(b) t_2时刻SQ的存储状态

(c) t_3时刻SQ的存储状态

图 3　单级缓存队列(SQ)的存储及请求处理过程

到达系统后即可进行处理，因而请求延迟也为 0；r_7^1 在 t_1 时刻到达后 t_2 时刻完成处理，故其对应的请求延迟为 1. 以此类推，ATBM 及 SQ 的延迟时间分别为：

ATBM 下请求延迟时间

$$d_1=6\times0+1\times2=2 \tag{2}$$

$$d_2=6\times0+6\times2+5\times3=27 \tag{3}$$

$$d_3=5\times0=0 \tag{4}$$

$$\begin{aligned} d_{\text{avg}}&=(d_1+d_2+d_3)/(C_1+C_2+C_3)/q\\ &=(2+27+0)/(1.1+2.9+0.9)/6\\ &=0.986 \end{aligned} \tag{5}$$

SQ 下请求延迟时间：

$$d_1=6\times0+1\times1=1 \tag{6}$$

$$d_2=5\times0+6\times1+6\times2=18 \tag{7}$$

$$d_3=5\times2=10 \tag{8}$$

$$\begin{aligned} d_{\text{avg}}&=(d_1+d_2+d_3)/(C_1+C_2+C_3)/q\\ &=(1+18+10)/(1.1+2.9+0.9)/6\\ &=0.986 \end{aligned} \tag{9}$$

与此同时，本文对两种缓存方法下不同延迟时间(0、1 s、2 s 以及 3 s)的分布情况进行了统计，统计结果如表 1、表 2、表 3 所示.

表 1　$\{C_1=1.1, C_2=2.9, C_3=0.9\}$访问 ATBM 及 SQ 请求延迟总体的分布情况

	$d=0$	$d=1$	$d=2$	$d=3$
ATBM	17	1	6	5
SQ	11	7	11	0

表 2　$\{C_1=1.1, C_2=2.9, C_3=0.9\}$访问 ATBM 时请求延迟的具体分布情况

ATBM	$d=0$	$d=1$	$d=2$	$d=3$
C_1	6	0	1	0
C_2	6	0	6	5
C_3	5	0	0	0

表 3　$\{C_1=1.1, C_2=2.9, C_3=0.9\}$访问 SQ 时请求延迟的具体分布情况

SQ	$d=0$	$d=1$	$d=2$	$d=3$
C_1	6	1	0	0
C_2	5	6	6	0
C_3	0	0	5	0

由式(5)、式(9)及表 1～表 3 可知，ATBM 在保证不增大平均请求延迟的基础上，优化了请求延迟的分布结构. 一方面，ATBM 能够使更多的并发量在“零延迟”的状态下完成响应，即 $d=0$；另一方面，ATBM 有效地降低了突发并发量对非突发并发量的影响，使得“高延迟”($d\geq2$)仅局限于突发并发量，而非突发并发量仍保持“低延迟”的处理效果. 观察表 2 及表 3 可知，当突发并发量 C_2 到达系统后，SQ 不能有效地对其进行隔离，使得随后到达的非突发并发量 C_3 被阻塞，进而导致的请求延迟增至 2 s；然而，ATBM 上层阻塞队列良好的“隔离效果”，使得“高延迟”仅发生在突发并发量 C_2，而非突发并发量 C_3 仍保持较低的延迟时间，进而改善了系统整体的服务效果.

与此同时,当突发并发量到达时,系统会识别、评估该突发,避免向已经发生严重"拥堵"的 VM 转发请求,然后采取例如负载均衡、调整 VM 配置、增加 VM 数量等方法应对该突发,以降低并发量的延迟,保证系统的 QoS.

3.2 基于 ATBM 的突发强度评估策略

ATBM 中核心参数队列级数 QL 和队列长度 q 的设置决定了 ATBM 的处理效果;其中,队列级数 QL 为缓存队列数与阻塞队列数之和,假设 $QL=5$ 时,系统则有 1 个缓存队列,4 个阻塞队列. 在队列长度 q 固定的情况下,QL 增大,系统单位时间内可存储的请求数增多,请求拒绝率下降,但其也会造成请求响应时间增长、资源利用率降低的后果. 同理,在 QL 固定不变时,q 过大,会导致上层阻塞队列利用率降低,失去阻塞队列对瞬时涌入的请求的隔离作用,使 VM 再次退化为传统的单一缓存模式. 然而,q 过小,会导致服务请求拒绝率增大;与此同时,由于缓存空间不足,VM 会出现大量请求被拒绝,而大量资源(如计算资源)被闲置的状况,从而导致资源利用率下降. 为了平衡 QoS 及资源利用率,我们将 QL 固定为 5,q 进行如下设置:

$$q=\theta\beta C/C_0,\ 0<\theta<(m+1) \tag{10}$$

其中 C 为 VM 当前的处理能力,C_0 为处理单个请求所需的资源量;假设 SLA 规定的请求最大响应时间为 $(m+1)\beta$,其中 $m>0$. 式(10)中使用 $\theta\beta$ 而未使用 $(m+1)\beta$,是因为 ATBM 中请求响应时间包括两部分内容:阻塞时间及缓存时间,两者分别为请求在阻塞队列及缓存队列中的等待时间,q 只需保证两部分时间之和小于 $(m+1)\beta$ 即可,无需要求缓存时间小于 $(m+1)\beta$. 与此同时,系统可以依据用户对请求平均响应时间的要求合理设置 θ;θ 越大,请求的平均响应时间越长,本文将 θ 设置为 1.

对并发量的突发强度进行评估时,需要同时考虑两方面因素:VM 当前的处理能力及并发量本身的变化情况[3]. 就 ATBM 而言,可使用其各级队列中请求的存储情况来描述 VM 当前的处理能力及并发量本身的变化情况,进而评估并发量的突发强度. 可由式(10)可知,当最大响应时间及单个请求的资源需求量固定时,VM 的处理能力越强,q 对应的值越大. 与此同时,当长度 q 固定后,ATBM 被占用的队列层数越多,说明单位时间内到达的请求数越多,并发量激增速度越快. 基于上述内容,本文如下评估方法:

$$b=\frac{\omega len(\text{cache})}{q}+(1-\omega)\sum_{i=1}^{n}\frac{len(\text{block}_i)}{q} \tag{11}$$

其中 $len(*)$ 表示队列 $*$ 当前存储的请求数,n 为 ATBM 中阻塞队列的层数,q 为 ATBM 中各级队列的长度,ω 为对应的权重.

4 最大可接受并发量评估算法 ACM

为了更细粒度、高精度地评估系统下一时刻的资源可用量,本文提出了一个基于 ATBM 的最大可接入并发量评估算法 ACM. ACM 对系统可用资源量进行了的具体化处理,将其转化为在保证 VM 不发生 SLA 违例(包括响应时间超时及请求违例率高于规定值)的前提下,VM 可接入的最大并发量数,然后得出系统的资源可用量. 在详细介绍 ACM 的计算过程前,我们先做出如下假设:

(1) VM 在 t 时刻的突发强度为 k,其对应的式(11)中的 ω 为 0;

(2) $[t,t+i\beta]$ 时段内进入 VM 的并发量为 $r(t+i)$,其中 $i=1,\cdots,m$;

(3) SLA 中规定的请求最大响应时间及违例率分别为 $(m+1)\beta$、α;

cache 中请求的响应时间均小于 β,故其不会发生 SLA 违例;但存于 block 中的请求却有可能发生违例. 这是因为后者中的请求当且仅当 cache 有空余空间时,才进行处理;若在 $m\beta$ 时段内 cache 均无可用空间,block 中的请求便会发生超时违例.

为了保证该 VM 的请求违例率低于 α,我们需要控制未来 $m\beta$ 时段内进入 VM 中的并发量,即限制单位时间 β 内进入 VM 的并发量不大于 cache 长度 q,以使 block 中部分请求可以在 $m\beta$ 内完成发生. 因此,$\{r(t+i)\mid i=1,\cdots,m\}$ 需要满足如下条件:

$$r(t+i)\le q,\ i=1,\cdots,m \tag{12}$$

那么 $[t+\beta,t+m\beta]$ 时段内,block 中可以处理的请求数为

$$q-r(t+\beta)+,\cdots,+q-r(t+m\beta)=mq-m\bar{r} \tag{13}$$

其中 $\bar{r}$ 表示平均可接受的并发量. 为了保证 $(m+1)\beta$ 内的请求超时违例率低于 SLA 中规定的 α,需要满足如下条件:

$$\frac{kq-(mq-m\bar{r})}{(k+1)q+m\bar{r}}<\alpha \tag{14}$$

式(14)中分母为 $[t,t+m\beta]$ 内到达的总并发量数目;分子为该时段内发生违例的并发量数. 将式(14)

整理后得出：

$$\bar{r}\leq q\frac{m+\alpha+(\alpha-1)k}{m(1-\alpha)} \tag{15}$$

由式(15)得到 $\bar{r}$ 后，便可求出系统资源可用量，计算方法见式(16)，其中 C_0 为单个请求对应的资源需求量.

$$C_{\text{available}}=\bar{r}\times C_0 \tag{16}$$

5　服务突发并发量应对策略 GMAC

5.1　改进的灰度预测模型 MGM

MGM 在继承 GM 对输入数据规律性要求较低的基础上，充分发掘预测残差中的欠预测信息，以降低预测结果的欠预测比例及规模，进而减少资源分配过程中的欠分配.

传统的 GM 在大部分情况下都可以保持较高精度，但当被预测点为异常点，例如并发量中的激增点及陡降点，其变化规律明显异于样本数据中的变化规律时，GM 便不可避免地产生预测误差. 假设在 $[t_i,t_{i+n-1}]$ 时段内用户并发量持续下降，GM 在 t_{i+n} 的预测值将延续这一递减规律. 然而，若 t_{i+n} 时刻的并发量激增，即并发量变化规律不同于先前的样本数据，GM 便会出现欠预测；反之，若并发量出现陡降，过预测便会发生. 如图 4 所示，t_8 时刻数据激增，GM 未能反映出该变化，预测结果出现欠预测；随后的陡降点使 GM 出现了过预测.

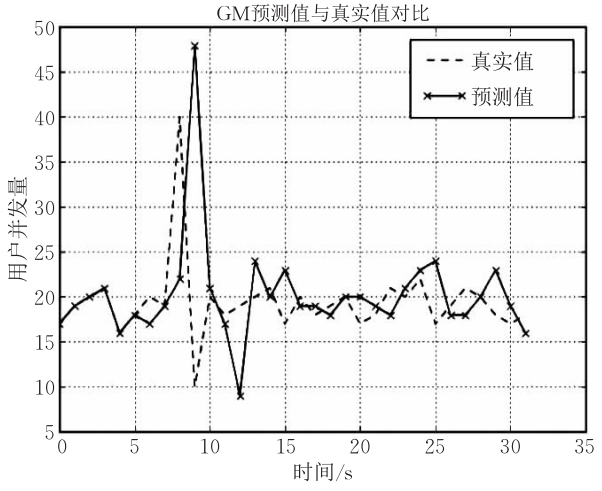


图 4　GM 对存在激增和陡降点数据的预测结果

为了降低 GM 的欠预测比例及规模，我们对其预测结果做出如下修正：

假设 $X_P^{(0)}=\{x_p^{(0)}(k+1),\cdots,x_p^{(0)}(k+n+1)\}$ $(n\geq 2)$ 为基于GM的并发量预测值，$X^{(0)}=\{x^{(0)}(k+1),\cdots,x^{(0)}(k+n)\}(n\geq 2)$ 为并发量真实值，对应的预测残差序列如式(17)所示. MGM 使用式(18)及式(19)提取残差序列中的欠预测信息，并利用该提取值对 GM 的预测结果进行调整，具体调整方法如式(20)及式(21)所示.

$$R=\{r_i\,|\,x_p^{(0)}(i)-x^{(0)}(i),k+1\leq i\leq k+n\} \tag{17}$$

$$R'=\{r_i'\,|\,k+1\leq i\leq k+n\} \tag{18}$$

$$r_i'=\begin{cases}0, & r_i<0\\ |r_i|, & \text{其它}\end{cases} \tag{19}$$

$$x_p^{(1)}(k+n+1)=x_p^{(0)}(k+n+1)+R_{\text{MGM}} \tag{20}$$

$$R_{\text{MGM}}=\frac{\sum_{i=k+1-n-Len}^{k+n}r_i'}{Len} \tag{21}$$

式(20)中 $x_p^{(1)}$ 为基于 MGM 的预测值，R_{MGM} 为过去一个预测窗口 Len 内欠预测结果的平均值. 基于 $x_p^{(1)}$ 可得到系统的资源需求量 C_{demand}，其中 C_0 为单个请求的资源需求量.

$$C_{\text{demand}}=x_p^{(1)}\times C_0 \tag{22}$$

基于式(16)及式(22)求得资源可用量及需求量后，便可制定相应的资源策略. 其中，资源策略分为两种：横向的 VM 数量调整及纵向的 VM 资源调整，前者调整的时效性优于后者[9]；因此，我们采用调整 VM 数量的方式应对服务突发并发量. 具体调整算法如式(23)及式(24)所示.

$$VM_{\text{adjust}}=\max\{-1,\lceil(C_{\text{demand}}-C_{\text{available}})/(\varepsilon q)\rceil\} \tag{23}$$

$$VM_{\text{demand}}=\max\{VM_{\min},VM_{\text{adjust}}+VM_{\text{current}}\} \tag{24}$$

其中，$\lceil\ \rceil$ 表示向上取整，VM_{adjust} 表示需要调整的 VM 数量，VM_{current} 为系统当前的 VM 数，VM_{demand} 为系统当前应分配的 VM 数，$VM_{\min}$ 为服务设定的最小 VM 数. 设置 $VM_{\min}$ 的目的是防止并发量持续降低后突然激增，导致系统 VM 数量无法从连续降低或长期保持较低值的状态瞬时切换到较高值，而引发 QoS 降级. ε 的取值范围不小于 1，它反映的是用户对请求响应时间的要求：ε 越大，VM_{adjust} 越小，系统中总的 VM 数越少，其对应请求响应时间越长，反之亦然；本文实验过程将 ε 设置为 1.

5.2　GMAC 的资源调整算法

为了从宏观角度上介绍 GMAC，本文先对其调整过程进行描述，具体内容如图 5 所示.

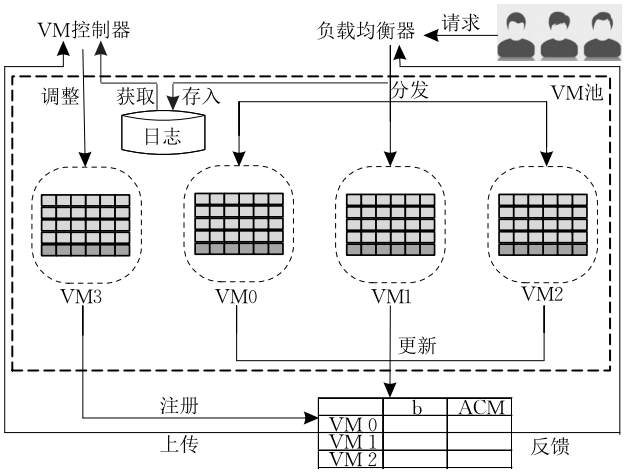


图 5 GAMA 的资源调整过程

过程 1. GMAC 资源调整过程.

(1)分发并发量

本文采用 ArA 算法[44]进行负载均衡,其原理为:突发强度较大时对并发量实施随机分发,突发强度较小时则进行贪心分发,即选择 ATBM 请求占用层级少、可用空间多的 VM 进行处理.与此同时,该负载均衡器需要将用户访问情况存入访问日志.

(2)更新突发强度评估表

为了保证 QoS 同时提高资源利用率,系统需要实时评估可用资源.具体而言,各 VM 需要基于式(11)及式(16)评估其当前的突发强度(b)及可用资源量(ACM),并将最新的评估结果上传至 VM 控制器,以供其制定相应的资源调整策略.

(3)制定并执行资源调整策略

VM 控制器将各 VM 的可用资源量加和,同时将用户访问日志中的历史数据输入 MGM 模型,求得资源需求量.最后,对比资源可用量及需求量,确定系统需要增减的 VM 数,并实施相应的增减操作.具体内容见算法 1 及算法 2.

(4)提交反馈信息

执行完以上操作后,系统中 VM 数量可能发生变化.其中,新部署的 VM 需要将其对应的状态信息(突发强度 b 及可接受的最大并发量 ACM)注册到突发强度表中;被销毁的 VM 则需从表中注销其对应的信息.突发强度评估表会将最新内容反馈给负载均衡器,为其分发并发量提供依据.

基于式(23)便可求得系统需要调整的 VM 数,其调整动作又可细分为增加 VM 与减少 VM.为了防止 VM 数量的频繁调整,当增加 VM 数量后短期内不能对 VM 进行移除操作;当且仅当移除 VM 的命令数累积超过某阈值时,才可执行该调整动作.GMAC 算法的具体过程如算法 1 所示.

算法 1. 基于 MGM 及 ACM 的 VM 调整算法.

输入:*data_W* //历史并发量数据
Len //MGM 预测窗口的大小
VMList //系统当前的 VM 列表
basicVMCount //最低 VM 数
q //ATBM 缓存队列的长度
CommandTimes //发出 VM 移除操作对应的命令数
ADJUSTTHRESHOLD //触发 VM 移除的阈值

输出:*optimalVMCount* //最优 VM 数量

```
1.  totalACM=0 //系统可接受的最大并发量
2.  VMCount=0
3.  FOR(VM:VMList)
4.  totalACM+=VM.ACM
5.  MGM_W=MGM(data_W,Len) //并发量预测值
6.  IF(MGM_W>totalACM)
    {
7.  VMCount=ceil{(MGM_W-totalACM)/q}
8.  createCreate(VMCount)
    //创建指定数量的 VM
9.  CommandTimes=0
    }
10. ELSE IF(MGM_W<totalACM)
    {
11. VMCount=ceil(MGM_W/q)
12. IF(VMList.size>VMCount
13. &CommandTimes>ADJUSTTHRESHOLD)
    {
14. CommandTimes++
15. VMList=remove()
    }
    }
16. optimalVMCount=VMList.size
17. RETURN optimalVMCount
```

算法 1 中 6 至 9 行分析得出系统当前处理能力不足,需要增加的 VM 数量,其中 *ceil*()函数表示向上取整.10 到 13 行评估发现系统处理能力过剩,需要减少的 VM 数量.为了避免频繁调整,同时防止 MGM 欠预测造成欠分配,在移除 VM 的过程,GMAC 会缓慢减少 VM,即每次操作仅移除一个 VM,且保证当前 VM 个数不低于系统规定的最低值 *basicVMCount*;与此同时,移除 VM 操作命令数 *CommandTimes* 积累超过指定阈值 *ADJUSTTHRESHOLD* 后,才可移除 VM.

VM 移除操作包括两个步骤:选择待移除的 VM、转移未处理完成的请求.首先要选取突发强度最低的 VM 作为待移除 VM,以最大限度地降低

移除操作对 QoS 的影响.然后要为未处理完成请求选择目标 VM,这直接影响着请求的处理效果.目前,选择目标 VM 的策略分为两种:贪心选择及随机选择.前者会选择目前最为空闲的 VM,后者则进行随机选择.本文采用后者作为目标 VM 选择策略,其原因在于随机选择无需再次评估系统 VM 空闲程度,降低了时间开销,提高了"再分配"过程的时效性;另一方面,随机选择可有效避免目标 VM 因缓存过多的"迁入请求",造成自身突发强度增大、QoS 降低,进而导致整体的 QoS 降级.该过程的实现细节如算法 2 所示.

算法 2.　VM 移除算法.

```
输入:VMList
输出:newVMList
1.  bursty=MAX_VALUE
2.  objVM=NULL
3.  FOR(VM:VMList)
4.      IF(VM.bursty<bursty)
    {
5.  objVM=VM
6.  bursty=VM.bursty
     }//寻找突发强度最低的 VM
7.  FOR (r:objVM.requestList)
    {
8.      rand=Random(VMList,objVM)
9.      migrate(r,rand)
    }//将目标 VM 上的请求随机
    //分发到其它 VM 中
10. VMList.remove(objVM)
11. newVMList=VMList
```

算法 2 反映的是 VM 移除过程,其主要包括 2 个步骤:选取突发强度最低的 VM 作为待移除的目标 VM 及随机分发该 VM 上未处理完成的请求到其它 VM 上,其分别对应算法 2 的 3 到 6 行及 7 到 9 行.其中,$Random(VMList,objVM)$ 表示从 $VMList$ 中随机选取除 $objVM$ 外的任意 VM.

就时间复杂度方面来说,GMAC 的时间消耗主要来源于 4 方面内容:MGM 算法对资源需求量的预测、ACM 模型对资源可用量的评估、ArA 算法实施的负载均衡以及基于并发量突发强度的 VM 增减操作.由 5.1 节 MGM 的预测原理可知,其对应的时间复杂度为 $O(n^2)$;GMAC 需要遍历系统内所有 VM 来评估资源可用量及突发强度,故 ACM、ArA 及 VM 增减操作的时间复杂度度为均 $O(n)$.综上可知,GMAC 的时间复杂度为 $O(n^2)$.

在分析空间复杂度时,本文也从上述 4 个方面入手:MGM 在预测过程中需要申请 win 个空间(预测窗口内的大小),来存放原始并发量数据;其中,win 是人为指定的常量,它与并发量规模无关,故而 MGM 的空间复杂度为 $O(1)$.如式(15)所示,ACM 基于 VM 当前的存储状态评估系统可接入的最大并发量,它只需要开辟一个空间来存放评估结果即可,所以其对应的空间复杂度为 $O(1)$.同时,系统需要创建、维护一张"突发强度评估表",来记录系统中各 VM 的突发强度,以供随后的负载均衡及 VM 增删操作使用.具体而言,当并发量规模增大时,系统创建的 VM 数量越增大,"突发强度评估表"记录的内容越多,其开辟的空间也便增大;因此,"突发强度评估表"的空间复杂度与并发量规模呈正相关,故其对应的空间复杂度为 $O(n)$.基于 ArA 算法的负载均衡策略以及基于并发量突发强度的 VM 增减操作,只需遍历该表便可得出对应的调整值,该过程只需开辟常数级别的辅助空间,故而两者对应的空间复杂度均为 $O(1)$.综上可知,GMAC 的空间复杂度为 $O(n)$.

6　实验结果及分析

6.1　实验数据及环境

本文采用由百度公司统计的 2015 年中以"淘宝商城"为关键字,在"百度搜索引擎"中进行搜索的并发量数据,即 2015 年"淘宝商城"对应的"百度指数",作为实验数据.如图 6 所示,该数据具有大规模性及高突发性这两大特征.在规模性方面,该数据的平均值为 74114 次/天,最小值为 24029 次/天、最大值高达 190578 次/天,最大值为最小值的 7.9 倍;其中,最小值发生在 2015 年 2 月 19 日(春节),最大值发生在 2015 年 11 月 11 日.就突发性而言,并发量数据从 11 月 10 日的 127141 次/天瞬时激增到 11 月 11 号的 190000 次/天,然后又陡降到不足 8500 次/天,11 月 11 日的并发量数据分别为 11 月 10 号及 12 号的 1.5 倍及 2.3 倍.

本文中所有的模型及算法均部署于开源项目 CloudSim[①],程序源码[②].实验所有内容均在配置为 Intel(R)Core(TM)i7-4590 CPU3.60 GHz、8 GB 内存、64 位 Windows7 操作系统的计算机中运行.

① http://www.cloudbus.org/cloudsim/
② https://github.com/JaneWuNEU/cloudsim

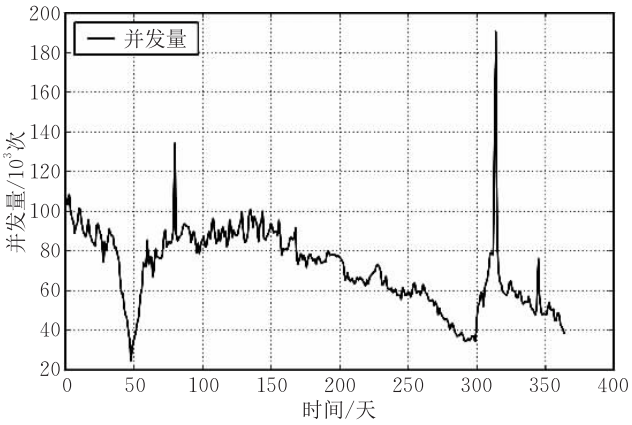


图 6 2015 年“淘宝”对应的百度指数的并发量数据

6.2 ATBM 和 SQ 下请求处理效果对比

为了进一步评估 ATBM 在改善 QoS 方面的有效性,本文令其与传统 VM 的单级缓存 SQ 就处理图 6 中并发量时,对应的请求违例率、拒绝率、平均响应时间以及系统资源利用率进行对比.其中 SLA 规定的服务最大响应时间为 0.55 s、最高违例率为 0.1.

影响 ATBM 及 SQ 缓存效果的因素有队列级数 QL 和队列长度 q;因此,文本将分析 ATBM 及 SQ 在不同队列级数、不同队列长度下的请求违例率、拒绝率、平均响应时间以及系统资源利用率,来对比、评估两者的请求处理效果.与此同时,为了避免负载均衡对实验结果的影响,该部分实验过程仅使用 1 台 VM,该 VM 的配置如下:1 核 CPU、1 GHz、内存大小为 521 MB.

6.2.1 不同队列级数下请求 QoS 对比

ATBM 的缓存队列长度 q,在设置过程中不可过长也不宜过短:q 过长会令 ATBM 退化为 SQ,进而无法就两者的突发并发量应对能力进行对比;q 过短,会使 ATBM 对应的拒绝率增大,而违例率及响应时间分别接近 0 及 0.1(请求的到达间隔),从而无法就后两方面内容与 SQ 进行比较.经实验验证得知,将 q 设置为 19 800,可取得较好的对比效果.由表 4 可知,队列级数越高,ATBM 较 SQ 在 QoS 方面的优势越大.当队列级数不大于 4 时,ATBM 较 SQ 具有更高的违例率,但此时 ATBM 的请求违例率远低于 SLA 中规定的 0.1,即 ATBM 未发生 SLA 违例;当队列级数超过 4 后,ATBM 的违例率便远低于 SQ,后者对应的违例率高于 0.1,即 SQ 发生 SLA 违例.随着队列级数的增大,VM 单位时间内可缓存的请求数增大.ATBM 对这些涌入的并发量进行了分级缓存,并为不同层级的请求赋予不同的处理优先级,有效避免了突发并发量对缓存队列的持续,保证了突发点过后的非突发并发量的及时处理,从而使 VM 的违例率被控制在了较低水平.

表 4 不同队列级数下 ATBM 和 SQ 的 QoS 对比

QL	违例率		拒绝率		平均响应时间/s	
	ATBM	SQ	ATBM	SQ	ATBM	SQ
$QL=2$	0.009	0	0.140	0.140	0.184	0.184
$QL=3$	0.016	0	0.132	0.132	0.261	0.261
$QL=4$	0.024	0	0.124	0.125	0.340	0.340
$QL=5$	0.031	0.126	0.118	0.118	0.418	0.419
$QL=6$	0.036	0.501	0.112	0.112	0.490	0.491

为了进一步对比 ATBM 和 SQ 的请求处理效果,本文对两者中请求的响应时间进行了更细粒度地划分,即分别统计请求响应时间 $d\in(d_i-0.11, d_i]$($d_i=0.11, 0.22, 0.33, 0.44, 0.55$)以及 $d>0.55$ 时的并发量占比,划分结果如表 5 所示.

表 5 不同队列级数下 ATBM 和 SQ 的请求延迟分布

QL	$0<d\leq0.11$		$0.11<d\leq0.22$		$0.22<d\leq0.33$		$0.33<d\leq0.44$		$0.44<d\leq0.55$		$d>0.55$	
	ATBM	SQ	ATBM	SQ	ATBM	SQ	ATBM	SQ	ATBM	SQ	ATBM	SQ
$QL=2$	0.833	0.268	0.019	0.556	0	0.037	0	0	0	0	0.009	0
$QL=3$	0.836	0.213	0.016	0.096	0	0.482	0	0.077	0	0	0.016	0
$QL=4$	0.837	0.179	0.015	0.075	0	0.080	0	0.441	0	0.101	0.024	0
$QL=5$	0.838	0.150	0.013	0.072	0	0.062	0	0.073	0	0.400	0.031	0.126
$QL=6$	0.844	0.140	0.008	0.060	0	0.061	0	0.064	0	0.061	0.036	0.501

ATBM 可以使 80%以上的请求在 0.11 s 内完成响应,且该并发量占比随着 QL 的增大而加大.然而,SQ 在 0.11 内完成响应的并发量占比却不足 30%,且该占比随着 QL 的增大而减小.另一方面,ATBM 在 $QL=i(i=2,3,4,5,6)$ 时 $d>0.55$ 对应的并发量占比,都不会超过 SLA 中规定的 0.1;当 SQ 在 $QL=5$ 时,$d>0.55$ 对应的占比便超过了 0.1,引发 QoS 降级.

就拒绝率和平均响应时间而言,无论队列级数为何值,两者都近似相等;随着队列级数的增大,两者的平均响应时间增长,但拒绝率在降低.当 $QL=2$ 时,ATBM 仅由一个缓存队列和一个阻塞队列构成.此时,系统的违例率不足 0.01,远低于 SLA 中规定的 0.1,平均响应时间也远小于 SLA 中要求的

0.55s，但系统拒绝率过高；同时，由于存储资源和计算资源不对等、不匹配，大量计算资源被闲置，造成资源利用率下降. 然而，当 $QL=5$ 时，虽然此时违例率与响应时间有所增大，但其并未超过 SLA 中规定的 0.1 及 0.55，而此时的拒绝率仅为 $QL=2$ 时的 84.3%. 因此，系统要合理设置队列级数，使拒绝率和平均响应时间处于较为平衡、合理的状态，以保证 VM 的服务效果.

综上可知，ATBM 和 SQ 相比，前者在保证请求拒绝率及平均响应时间不增大的同时，优化了请求延迟的分布结构. ATBM 能使更多的请求在较短的延迟内完成响应，同时其将“高延迟”的并发量占比也控制在了合理范围，从而有效避免了 SLA 违例.

6.2.2　不同队列级数下资源利用率对比

本文使用 VM 各级队列在单位时间内的平均占用率代表资源利用率，其计算方法如式(25)所示.

$$UR=\frac{1}{Len}\sum_{i=1}^{Len}\frac{store_i}{q} \tag{25}$$

UR 表示队列的占用率，包括 ATBM 下的 cache 队列和 $block_i$ ($i=1,\cdots,4$) 队列以及 SQ 下的 cache 和 pause 队列；Len 表示评估窗口的大小，本实验中 Len 的取值为 365；$store_i$ 表示第 i 时刻队列存储的并发量数；q 表示队列的长度，本实验将其设置为 19800. SQ 中 cache 及 pause 的长度分别为 19800 及 59400，即单位时间内最多可缓存 19800 及 59400 个请求. 表 6 中最后一列 $ATBM\_AVG$ 为 ATBM 中 $QL=i(i=2,3,4,5,6)$ 时，阻塞队列占用率的平均值，其计算方法如式(26)所示：

$$ATBM\_AVG_{QL=i}=\sum_{j=1}^{i-1}\frac{UR_{block_j}}{i-1} \tag{26}$$

$ATBM\_AVG_{QL=i}$ 为队列级数 $QL=i$ 时 ATBM 阻塞队列占用率的平均值；UR_{block_j} 表示 ATBM 中阻塞队列 $block_j$ 的占用率. 由最终的计算结果可知，当队列级数 $QL<6$ 时，$ATBM\_AVG$ 会随着队列级数的增大而增大；但是，当 $QL=6$ 时 $ATBM\_AVG$ 反而会降低. 这是因为 $QL=5$ 及 $q=19800$ 足以保证 VM 的处理效果，如果再增大队列级数，一方面会增大请求的平均响应时间，另一方会导致资源闲置，即 ATBM 顶层的阻塞队列长期处于“低载”状态，进而造成资源利用率降低，产生不必要的资源开销.

表 6　不同队列级数下 ATBM 和 SQ 中各级队列的资源利用率

QL	cache		$block_1$		$block_2$		$block_3$		$block_4$		$block_5$		ATBM_AVG
	ATBM	SQ	ATBM	SQ	ATBM	SQ	ATBM	SQ	ATBM	SQ	ATBM	SQ	ATBM
$QL=2$	0.919	0.919	0.620	0.620	0	N/A	0	N/A	0	N/A	0	N/A	0.620
$QL=3$	0.927	0.927	0.690	0.639	0.589	N/A	0	N/A	0	N/A	0	N/A	0.640
$QL=4$	0.935	0.935	0.736	0.655	0.659	N/A	0.569	N/A	0	N/A	0	N/A	0.655
$QL=5$	0.943	0.943	0.776	0.663	0.701	N/A	0.634	N/A	0.542	N/A	0	N/A	0.663
$QL=6$	0.948	0.948	0.798	0.659	0.732	N/A	0.664	N/A	0.587	N/A	0.514	N/A	0.659

由表 6 可知，ATBM 和 SQ 的资源利用率近似相等. 具体而言，ATBM 及 SQ 中缓存队列 cache 的占用率在 $QL=i(i=2,3,4,5,6)$ 时均相等，且占用率会随着队列级数 QL 的增大而增大. 队列级数的增加使得 VM 的请求拒绝率降低，单位时间内可存储的请求数增多；在队列长度 q 及评估窗口 Len 不变的情况下，存储的请求数 $store_i$ 增大，会使占用率 UR 增大. 与此同时，ATBM 各级缓存队列的平均占用率 $ATBM\_AVG$ 与 SQ 中缓存队列 pause 的占用率也近似相等. 因此，ATBM 在改善 VM 中请求处理效果的同时，也能保证 VM 的资源利用率.

6.2.3　不同对队列长度下 QoS 对比

本实验首先将 QL 设置为 5，cache 队列的长度固定为 19800，然后将阻塞队列长度 q 分别为 16500、18000、19800、21000 以及 22500，它们分别对应并发量 20%分位数、30%分位数、43%分位数、45%分位数以及 50%分位数；最后评估不同队列长度 q 下 VM 对应的请求违例率、拒绝率、平均响应时间以及更细粒度的响应时间分布情况，实验结果如表 7 及表 8 所示.

由表 7 可知，当队列长度 q 低于 18000 时，ATBM 的违例率和响应时间高于 SQ，两者的拒绝率近似相等，此时两者均未发生 SLA 违例. 然而，当队列长度高于 18000 时，ATBM 的 QoS 则优于 SQ. 具体而言，当 $q>18000$ 时，ATBM 的平均响应

表 7　不同队列长度下 ATBM 和 SQ 的 QoS 对比

q	违例率		拒绝率		平均响应时间/s	
	ATBM	SQ	ATBM	SQ	ATBM	SQ
$q=16500$	0.027	0.015	0.122	0.123	0.369	0.354
$q=18000$	0.029	0.063	0.119	0.120	0.394	0.384
$q=19800$	0.031	0.126	0.118	0.118	0.418	0.419
$q=21000$	0.032	0.242	0.117	0.116	0.430	0.441
$q=22500$	0.034	0.383	0.115	0.114	0.459	0.468

表 8 不同队列长度下 ATBM 和 SQ 的延迟分布

q	$0<d\leq0.11$		$0.11<d\leq0.22$		$0.22<d\leq0.33$		$0.33<d\leq0.44$		$0.44<d\leq0.55$		$d>0.55$	
	ATBM	SQ	ATBM	SQ	ATBM	SQ	ATBM	SQ	ATBM	SQ	ATBM	SQ
$q=16500$	0.798	0.174	0.053	0.074	0	0.071	0	0.386	0	0.156	0.027	0.015
$q=18000$	0.800	0.161	0.052	0.074	0	0.064	0	0.247	0	0.271	0.029	0.063
$q=19800$	0.838	0.150	0.013	0.072	0	0.062	0	0.073	0	0.400	0.031	0.126
$q=21000$	0.851	0.146	0	0.067	0	0.064	0	0.064	0	0.299	0.032	0.242
$q=22500$	0.804	0.143	0.047	0.064	0	0.062	0	0.065	0	0.170	0.034	0.383

时间低于 SQ，拒绝率两者近似持平；但是，ATBM 的违例率却远低于 SQ，前者的违例率最低可降至 SQ 的 10%. 当队列 q 增大时，ATBM 的违例率保持较为缓慢、平稳地增长，而 SQ 的违例率会随队列长度的增大发生激烈变化. SQ 在 $q=19800$ 时便了发生 SLA 违例，当 $q>19800$ 时，违例情况会进一步加重；与此同时，SQ 中请求的平均响应时间也在增长，进而导致系统的 QoS 持续恶化.

本文就不同队列下两者延迟的分布情况进行了统计，具体结果如表 8 所示：ATBM 能保证 80%以上的请求在 0.11s 内完成响应；而 SQ 却不足 18%，且该并发量占比会随着队列长度的增大而减少. 其中的原因在于缓存队列 q 增大时，SQ 中 pause 队列的长度也会增大，即 SQ 可存储的请求数增大. 当突发并发量到达 VM 时，大量的请求会阻塞到 pause 队列中；当突发过后并发量恢复到非突发状态时，由于 pause 队列早已被突发并发量占用，最新到达的请求只能在 pause 队尾进行等待.

此时，队列长度 q 越大，pause 队列越长，非突发并发量占用的位置越靠后. 在系统处理能力不变的情况下，pause 中请求“移动”到 cache 队列的速度固定. 在“移动速度”不变的情况下，队列越长，队尾请求移动到队头所需的时间越长，其出现响应超时的概率越大，系统的违例率也便增加. 然而，ATBM 的“隔离机制”能有效降低突发并发量对非突发并发量的影响，其能保证非突发并发量在到达时具有更高的处理“优先级”，而不是在阻塞队列的尾部长期“等待”，进而缩短了非突发并发量的响应时间，降低了系统的违例率. 与此同时，“优先级”的变更虽然会导致突发并发量的响应时间增长，但实验证明超时的并发量占比远低于 SLA 的规定值，系统此时并未产生 SLA 违例. 因此，ATBM 较 SQ 更有效地改善系统的服务效果.

6.2.4 不同对队列长度下资源利用率对比

本文将分别基于式(27)和式(28)计算 ATBM 及 SQ 内各队列的占用率，进而对比、评估两者在队列级数 QL 固定、队列长度 q 变动的情况下的资源利用率.

$$UR_{\text{block}_j}=\frac{1}{Len}\sum_{k=1}^{Len}\frac{store_k}{i} \tag{27}$$

$$UR_{\text{pause}}=\frac{1}{Len}\sum_{k=1}^{Len}\frac{store_k}{(QL-1)i} \tag{28}$$

式(27)中的 UR_{block_j} 表示队列长度 $q=i$ 时，阻塞队列 block_j 的占用率，$store_k$ 表示队列 k 时刻存储的请求数；式(28)中的 UR_{pause} 表示队列在队列长度为 i 时，SQ 中 pause 队列的占用率.

基于上述内容求得的各队列的占用率如表 9 所示：ATBM 中 cache 队列、block_2 队列以及 SQ 的 cache 和 pause 队列的占用率会随着 q 的增大而提高；block_4 的占用率则在 q 增大时逐步降低；block_1 及 block_3 的占用率在 q 逐步增大的过程中，呈现出“先增后减”的变化趋势. 队列长度 q 的增大降低了系统的拒绝率，使得系统可处理的请求数增多. 若此时 VM 的处理能力固定，ATBM 中 cache 队列的占用率(资源利用率)则会随着处理请求总数的增多而增大. 然而，若队列长度 q 过大，以致 $QL\times q(QL<5)$ 的存储空间足以应对峰值阶段的并发量时，ATBM 上层的阻塞队列 block_4 便会出现“低载”的现象，进而引起资源利用率降低.

表 9 不同队列长度下 ATBM 和 SQ 的资源利用率

q	cache		block_1		block_2		block_3		block_4		ATBM_AVG
	ATBM	SQ	ATBM	SQ	ATBM	SQ	ATBM	SQ	ATBM	SQ	ATBM
$q=16500$	0.940	0.937	0.759	0.624	0.693	N/A	0.634	N/A	0.555	N/A	0.660
$q=18000$	0.942	0.940	0.768	0.646	0.699	N/A	0.636	N/A	0.553	N/A	0.664
$q=19800$	0.945	0.943	0.776	0.663	0.701	N/A	0.634	N/A	0.542	N/A	0.663
$q=21000$	0.946	0.945	0.767	0.665	0.703	N/A	0.630	N/A	0.535	N/A	0.662
$q=22500$	0.949	0.947	0.760	0.666	0.707	N/A	0.627	N/A	0.530	N/A	0.663

6.3 ACM 计算结果准确性评估

ACM 综合考虑 VM 当前的缓存情况、资源配置及 SLA 规定后，得出了 VM 在不发生 SLA 违例的情况下可接受的最大用户并发量. 为了验证 ACM 计算

结果的准确性，本文将 ACM 的计算结果依次扩大到 $\sigma=\{1.0,1.05,1.1,1.2\}$ 倍，然后将扩大后的并发量加载到以 ATBM 为缓存机制的 VM 中，最后通过对比不同并发量下的 QoS，评估 ACM 计算结果的准确性. 该过程的具体内容如下：

(1) 假设 t_0 时刻并发量为 110，其中 ATBM 各级队列长度为 27；

(2) $\{t_i \mid i=1,\cdots,19\}$ 时刻对应的并发量均为基于 ACM 计算求得的最优值 $\{r^i \mid i=1,\cdots,19\}$，其中 $r^0=110$；

(3) 令 $\{r^i\}$ 分别乘以 $\sigma=\{1.0,1.05,1.1,1.2\}$ 得到 $\{r_k^i \mid k=0,\cdots,3;i=1,\cdots,19;r_k^0=110\}$；

$$r_k^i=\begin{cases}110, & i=0\\ r^i\times\sigma[k], & 1\le i\le 19\end{cases} \tag{29}$$

(4) 分析 ATBM 在处理 $\{r_k^i \mid k=0,\cdots,3\}$ 时对应的 QoS.

其中 r^i 的计算结果如图 7 所示，初始点的并发量 r^o 造成 ATBM 拥堵，为了保证该部分并发量的服务效果，下一时间周期 VM 接入的用户并发量会减少，因而 r^1 和 r^2 对应的并发量下降到 14.

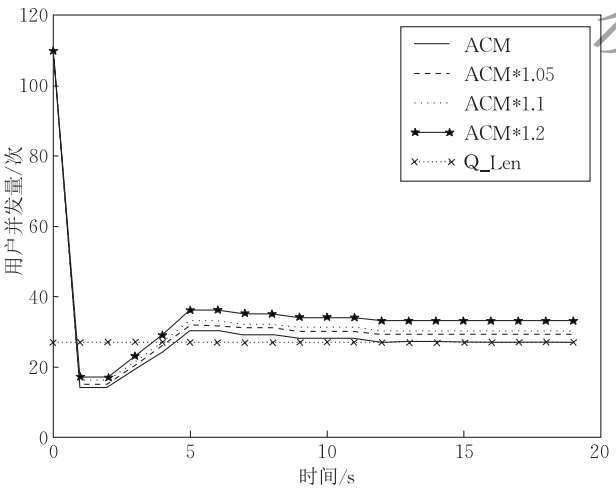


图 7　基于 ACM 计算求得的最佳用户并发量数据

表 10 反映的是 VM 在不同 $\{r_k^i\}$ 下的 QoS，r^i 扩大的倍数越大，VM 的违例率、拒绝率越大，平均响应时间越长. 其中，r_0^i 保证了最优的 QoS，即不存在任何 SLA 违例；将 r_0^i 扩大 1.05 倍得到 r_1^i，对应的请求违例率及平均响应时间便分别增加到原来的 2.5 倍及 1.36 倍；其中，当将 r^i 增大到 1.2 倍后，VM 便会发生 SLA 违例. 由此可知，当 VM 接入 ACM 求得的并发量 r^i 时，可保证最优的 QoS.

表 10　VM 处理 $\{r_k^i\}$ 对应的 QoS

$\{r_k^i\}$	违例率	拒绝率	平均响应时间/s
r_0^i	0.022	0	0.119
r_1^i	0.055	0	0.162
r_2^i	0.070	0	0.184
r_3^i	0.132	0	0.258

6.4　MGM 预测效果分析

在分析 GMAC 突发并发量应对效果之前，首先对其核心组成元素 MGM 在降低欠预测方面的有效性进行评估. 本文选择 8 个主流的并发量预测算法：传统的 GM、ARIMA、随机森林(Random Forest)、梯度提升决策树(GBDT)、线性回归(Linear Regression)、支持向量回归(SVR_lr、SVR_rbf)以及神经网络(NN)与 MGM 进行对比并评估各自的欠预测情况，其中 SVR_lr 及 SVR_rbf 分别表示 SVR 以 lr 及 rbf 作为核函数进行预测. 经实验验证，将 RandomForest 和 GBDT 中树的个数置为 10，GM、LinearRegression、MGM 的预测窗口[①]分别调整为 10、15、10，SVR_lr 及 SVR_rbf 中的惩罚因子固定为 1e3，NN 由 2 层全连接层构成、神经元个数分别为 8、1，激活函数为 relu，优化算法为 adam，迭代为 200 次时，可取得较好的预测效果.

本文使用的实验数据如图 6 所示，图中并发量数据在第 49 天[②]时陡降，然后迎来缓慢回升，并于 81 天发生小规模突发；并发量在第 315 天(2015 年 11 月 11 日)达到峰值，该时刻点之前的 50 天以及随后的 40 天，并发量均呈现出递减的趋势.

评估过程中欠预测比例及欠预测规模的计算方法如式(30)、式(31)所示.

$$P_{U_{\text{ratio}}}=\frac{count((w_{rj}-w_{pj})>0)}{Len},\ 1\le j\le Len \tag{30}$$

$$P_{U_{\text{volume}}}=\max\left\{\frac{\max\{w_{rj}-w_{pj},0\}}{w_{rj}}\ \middle|\ 1\le j\le Len\right\} \tag{31}$$

其中 w_{rj} 和 w_{pj} 分别为第 j 时刻并发量的真实值及预测值；Len 为预测区间的长度，本实验中 Len 取 365，即为图 6 所示用户并发量的数据规模；$count((w_{rj}-w_{pj})>0)$ 统计 Len 内 $(w_{rj}-w_{pj})>0$ 的总个数，即 Len 内欠预测的个数；式(31)则计算 Len 内欠预测规模. 计算结果如表 11 所示.

表 11 中 SUM 列统计的是欠预测比例及欠预测规模之和；SUM 对应的值越小，说明算法对欠预测的修正效果越佳. 其中，MGM 的对应的 SUM 值最

① 基于预测窗口内的数据对下一时刻点的并发量进行预测，例如当 MGM 的预测窗口设置为 10 时，算法会基于 $\{t_i \mid i=0,\cdots,9\}$ 内数据预测 t_{10} 时刻的并发量.

② 2015 年第 49 天为 2 月 19 日—春节，“淘宝”于该天停止发货，故“淘宝”对应的百度指数陡降.

表 11 各算法的欠预测比例及规模

	比例	规模	SUM	时间开销/s
MGM	0.36	0.37	0.73	0.712
ARIMA	0.51	0.49	1.00	14536.416
RF	0.39	0.59	0.98	5.482
GBDT	0.35	0.46	0.81	4.914
GM	0.44	0.51	0.95	0.584
LR	0.45	0.62	1.07	1.115
SVR_lr	0.48	0.56	1.04	0.814
SVR_rbf	0.48	0.65	1.13	0.807
NN	0.36	0.58	0.94	43.652

注:表中 RF 表示 RandomForest, LR 为 LinearRgeressio.

小,其仅为 SVR_rbf 的 64.6%及 GBDT 的 90.1%. 总体而言,MGM,GBDT 和 NN 的预测效果优于 ARIMA、LinearRegression、SVR_lr、SVR_rbf. ARIMA 要求并发量数据经过差分后可转换为平稳序列,但突发点的存在导致上述条件无法满足,因而造成预测精度降低;LinearRegression 试图用线性关系拟合并发量变化趋势,但正如图 6 中的并发量变化过程所示,其并不满足线性变化;SVR_lr、SVR_rbf 将并发量数据由低维转为高维,并希望在高维中找到并发量的变化规律,但是存在严重突发点的并发量数据由低维转到高维后,其可能仍不存在某种特定规律,因而 SVR_lr、SVR_rbf 的效果也便无法保证. GM 通过构建时间和并发量的函数关系进行预测,然而突发点会导致该函数关系的准确度降低,进而造成预测精度降低. MGM 充分挖掘预测结果中的欠预测信息来确定修正值,然后将该修正值与 GM 的预测结果相加,进而获得更低的欠预测规模.

在完成对各算法预测精度的评估后,本文对它们预测过程的时间开销也进行了分析,即算法运行一次的时间开销. 为了避免计算误差,表 11 中"时间开销"列所有内容都是 3 次运行结果的平均值.

由表 11 可知,ARIMA 的时间开销最大,其原因在于 ARIMA 在拟合参数 p,q 的过程中,需要进行大量的矩阵运算,从而造成时间开销增大. RandomForest、GBDT 及 NN 较 GM、Linear Regression 保持了更高的预测精度,但是前者的时间开销也远高于 GM 和 Linear Regression. 这是因为 Random Forest、GBDT 及 NN 内部由大量的子模型组成,它们需要大量的时间来构建、训练这些子模型(如神经元、决策树);GM 和 LR 仅由时间及并发量这单一维度的对应关系构成,确定该对应关系所需的时间远低于前者. MGM 在预测过程中需要统计预测窗口内的欠预测信息,因此它的时间开销略高于 GM,但其远低于其它 7 个预测方法.

综上可知,MGM 在欠预测比例、欠预测规模以及预测过程的时间开销均低于目前主流的预测算法,因此 MGM 具有更高的准确性以及更优的时效性.

6.5 GMAC 突发应对效果及资源利用率评估

为了表明 GMAC 在突发并发量应对方面的有效性,本文分别采用:ATBM+PEAK、ATBM+AVERAGE、SQ+PEAK、SQ+AVERAGE 以及 GMAC 作为资源调整策略,评估它们对图 6 中并发量的应对效果及资源利用率. 其中 PEAK、AVERAGE 分别表示系统基于并发量峰值及均值预留资源,就本实验而言,分别设置为 2 台 VM 及 1 台 VM,上述 VM 的配置均为:1 核 CPU、1 GHz、内存大小为 521 MB. ATBM、SQ 表示 VM 采用 ATBM 或 SQ 作为缓存机制,本实验中 ATBM 的队列级数 $QL=5$(1 级缓存队列、4 级阻塞队列),各级队列的长度均为 19800;SQ 中的 cache 队列长度为 19800,pause 队列的长度为 $19800\times(QL-1)=79200$. 因此,ATBM+PEAK 表示系统中所有 VM 都采用 ATBM 作为并发量请求缓存机制,同时基于峰值并发量进行资源预留. 就应对效果方面,本文对上述五种应对策略下的请求违例率、拒绝率及平均响应时间进行分析;资源利用率则分析 VM 各级队列在单位时间内平均的占用率,其计算方法见式(32)、式(33)以及式(34).

表 12 反映的是 5 种资源调整策略下 QoS 的对比结果,ATBM+PEAK 及 SQ+PEAK 均基于并发量峰值预留资源,因此其不存在请求违例及请求拒绝. 当基于并发量均值预留资源时,ATBM+AVERAGE 中的违例率远低于 SLA 中规定的 0.1,但 SQ+AVERAGE 的违例率高于 SLA 规定的 0.1,此时前者的违例率仅为后者的 24.6%;平均响应时间及拒绝率方面,两者近似相等. 基于上述内容可得出,在相同的资源预留策略中,VM 采用 ATBM 作为缓存机制较 SQ 具有更优的 QoS,从而再次证明 ATBM 在改善 QoS 方面的有效性. 但是,对比 GMAC 与 ATBM+AVERAGE 及 ATBM+PEAK 可知,GMAC 在保证系统不发生 SLA 违例的同时,能有效降低系统的拒绝率及平均响应时间,进而改善 QoS.

表 12 各资源调整策略下 QoS 对比

Type	违例率	拒绝率	平均响应时间/s
ATBM+PEAK	0	0	0.110
SQ+PEAK	0	0	0.110
ATBM+AVERAGE	0.031	0.118	0.419
SQ+AVERAGE	0.126	0.118	0.419
GMAC	0.036	0.098	0.383

在全面评估各算法的突发应对效果后，本文就5种策略的资源利用率也进行了分析，分析结果见表13. 表13中 AVG 列对应ATBM阻塞队列的占用率及SQ中pause队列的占用率. 具体而言，当VM的缓存模式为ATBM时，AVG 的计算方法如下：

$$AVG=\frac{1}{size}\sum_{i=1}^{size}ATBM\_AVG_i \tag{32}$$

其中，$size$ 表示VM总数，ATBM+PEAK对应的 $size=2$；当调整策略为ATBM+AVERAGE时，$size=1$. $ATBM\_AVG_i$ 则表示 VM_i 中阻塞队列的平均利用率(此时 QL 固定为5)，其具体计算过程见式(26). 当VM的缓存模式为SQ时，AVG 的计算方法如下：

$$AVG=\frac{1}{size}\sum_{i=1}^{size}UR_{pause_i} \tag{33}$$

表13　不同资源调整策略下的资源利用率

	cache	$block_1$	$block_2$	$block_3$	$block_4$	AVG	SUM
ATBM+PEAK	0.537	0.002	0	0	0	0.002	0.539
SQ+PEAK	0.535	0	N/A	N/A	N/A	0	0.535
ATBM+AVERAGE	0.945	0.776	0.701	0.634	0.542	0.663	1.608
SQ+AVERAGE	0.943	0.663	N/A	N/A	N/A	0.663	1.606
GMAC	0.705	0.714	0.640	0.574	0.488	0.604	1.309

资源调整策略SQ+PEAK、SQ+AVERAGE对应的 $size$ 分别为2、1；UR_{pause_i} 为SQ中pause队列的利用率，在计算应对策略GMAC对应的AVG值时，本文采用如下计算方法：

$$AVG=\frac{\sum_{j=1}^{size}\left(\sum_{k=1}^{4}UR_{\mathrm{block}_{k,j}}\right)/4}{count\left(\left(\sum_{i=1}^{4}UR_{\mathrm{block}_i}\right)/4>\rho\right)} \tag{34}$$

其中 $UR_{\mathrm{block}_{k,j}}$ 表示 VM_j 中 $block_k$ 队列的占用率，$\sum_{k=1}^{4}UR_{\mathrm{block}_{k,j}}/4$ 计算的是 VM_j 阻塞队列的平均占用率；$count\left(\left(\sum_{i=1}^{4}UR_{\mathrm{block}_i}\right)/4>\rho\right)$ 统计的是 $\{VM_j \mid j=1,\cdots,size\}$ 中阻塞队列平均占用率高于 ρ 的VM数量，本文将 ρ 设置为0.3.

GMAC下AVG的计算方法与其它策略有所不同，GMAC对式(32)及式(33)中的分母进行了调整，不再直接使用 $size$. 为了更清楚的解释其原因，本文对GMAC应对突发并发量过程中，动态调整的各VM的资源利用率进行了统计，具体结果见表14.

表14　GMAC调整过程中VM的资源利用率

	cache	$block_1$	$block_2$	$block_3$	$block_4$
VM0	0.754	0	0	0	0
VM1	0.935	0.714	0.640	0.574	0.488
VM2	0.684	0	0	0	0
VM3	0.447	0	0	0	0

由表14可知，GMAC处理并发量的过程中，共创建过4台VM，其中VM1承担了更多的处理任务，其它VM则只在并发量达到峰值时被创建，当并发量降低且系统持续发出销毁VM的命令时，突发强度最低的VM被系统销毁. 这些被“紧急创建”同时又被“及时销毁”的VM，由于服务时间短、处理请求数少，故而它们内部阻塞队列的占用率较低且可能出现大规模为0的情况. AVG 反映的是“常态”下VM的阻塞队列的利用率；如果那些存在时间极短、占用率极低的VM也加入 AVG 的评估过程，会导致评估结果出现严重偏差. 为此，本文引入一个控制变量 ρ，来筛选可加入评估过程的VM，即当且仅当VM阻塞队列的平均占用率高于 ρ 时，该VM才能加入评估过程；其中，ρ 越大，表示 AVG 评估过程对系统中VM“生存时间”要求越长，反之亦然. 但 ρ 不可过大，否则评估过程会“退化”成资源预留式系统资源利用率的评估，进而无法体现GMAC基于并发量变化规律动态、弹性地调整资源的特性.

如表13所示，本文将cache列及 AVG 列的对应值加和得到 SUM 列. 由 SUM 列可知，GMAC的资源利用率远高于基于峰值并发量进行资源预留的ATBM+PEAK及SQ+PEAK，前者的资源利用率是后两者的2.5倍. 其原因在于突发并发量的低频性及激增性，使得基于峰值预留的资源在大部分时间都处于闲置状态，因此造成其对应的资源利用率低于依据并发量进行实时、动态调整的GMAC的资源利用率.

GMAC的资源利用率较ATBM+AVERAGE及SQ+AVERAGE有所降低，前者的资源利用率分别为ATBM+AVERAGE及SQ+AVERAGE的81.7%及83.9%. 正如表14所示，GMAC为了降低系统的拒绝率及平均响应时间会动态增减VM，这些被“临时”创建、“生存时间”较短的VM，通常具有较低的资源利用率，进而导致系统整体的资源利用率降低. 但是，GMAC的VM“动态伸缩”机制，有效地改善系统的QoS；其能使系统的拒绝率及平均响应时间降至预留式资源调整策略

(ATBM+AVERAGE 及 SQ+AVERAGE)的 83.1% 及 91.4%,同时将违例率也控制在 SLA 规定的范围之内.

7 结 论

在应对云环境下突发并发量时,基于阈值设定的被动式应对策略存在时效性差,同时基于统计模型的主动式资源预留存在资源利用率低的问题,其会引发系统 QoS 下降及 SLA 违例.为了解决上述问题,本文提出了一个基于多级队列的 VM 缓存机制 ATBM 及一个综合的主动式突发并发量应对策略 GMAC.ATBM 将 VM 传统的单级缓存队列切分为缓存队列及阻塞队列,使用前者实时缓存并发量,同时使用后者隔离、阻塞瞬时涌入的突发并发量,从而有效避免了突发并发量对缓存队列的持续拥堵,改善了系统的 QoS.GMAC 由改进的灰度模型 MGM 及基于 ATBM 的最大可接受并发量评估算法 ACM 构成.其中,MGM 模型在继承 GM 对数据变化规律高鲁棒性以及预测过程高时效性的基础上,充分挖掘预测结果中的欠预测信息,有效地降低了资源需求量预测结果的欠预测比例及欠预测规模,提升了应对策略的有效性.ACM 则全面融合 VM 初始配置、请求缓存情况及 SLA 规定,就资源可用量实施评估,解决了目前可用量评估模型粗粒度、低精度的问题,提高了应对策略的准确性.GMAC 在应对突发并发量的过程中,首先利用 MGM 预测资源需求量,然后调用 ACM 评估系统的资源可用量,最后综合分析需求量及可用量,确定资源调整量.

GMAC 通过提前制定、执行资源调整策略,使得系统在并发量到达之前便可完成资源分配及部署,从而保证了应对策略的时效性.另一方面,GMAC 根据实时的并发量动态、弹性地调整 VM 数量,提高了系统的资源利用率.实验表明结果,在 VM 配置情况及并发量访问规模相同的情况下,VM 采用 ATBM 作为缓存机制时的请求违例率仅为 SQ 下请求违例率的 24.6%;与此同时,MGM 的欠预测比例及欠预测规模可降至现有预测算法欠预测比例及规模的 70.6% 及 56.9%.GMAC 则能在保证系统不发生 SLA 违例的前提下,将现有资源调整策略的资源利用率提高 1.45 倍,平均响应时间和拒绝率降低 8.6% 及 16.9%.

目前,GMAC 中的可用资源评估模型 ACM 在求解 VM 最大可接入并发量的过程中,有两方面的假设条件:(1)默认并发量为同类型并发量,即并发量的各类资源需求量均相同;(2)基于 VM 当前的 CPU 处理能力,确定可接入的并发量数量.在未来的工作中,我们将进行进一步的细化,即将并发量划分为计算密集型及 I/O 密集型,然后综合考虑 VM 的 CPU 处理能力、内存大小、带宽等多方面因素,确定可接入的不同类型并发量的数量,进而做出更适配的突发并发量应对策略.

参 考 文 献

[1] Gong Z, Gu X, Wilkes J. PRESS: PRedictive Elastic ReSource scaling for cloud systems//Proceedings of the IEEE International Conference on Network and Service Management. Ontario, Canada, 2011: 9-16

[2] Qiu X, Dai Y, Xiang Y, Xing L. A hierarchical correlation model for evaluating reliability, performance, and power consumption of a cloud service. IEEE Transactions on Systems Man and Cybernetics Systems, 2016, 46(3): 401-412

[3] Ali-Eldin A, Seleznjev O, Sjöstedt-de Luna S, et al. Measuring cloud workload burstiness//Proceedings of the IEEE/ACM, International Conference on Utility and Cloud Computing. London, UK, 2014: 566-572

[4] Mi N, Casale G, Cherkasova L, Smirni E. Burstiness in multi-tier applications: Symptoms, causes, and new models//Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware. New York, USA, 2008: 265-286

[5] Mohamed M, Amziani M, Belaïd D, et al. An autonomic approach to manage elasticity of business processes in the cloud. Future Generation Computer Systems, 2015, 50(C): 49-61

[6] Shen Z, Subbiah S, Gu X, Wilkes J. CloudScale: Elastic resource scaling for multi-tenant cloud systems//Proceedings of the ACM Symposium on Cloud Computing. New York, USA, 2011: 5

[7] Ali-Eldin A, Tordsson J, Elmroth E. An adaptive hybrid elasticity controller for cloud infrastructures//Proceedings of the IEEE Network Operations and Management Symposium. HI, USA, 2012: 204-212

[8] Hasan M Z, Magana E, Clemm A, Tucker L. Integrated and autonomic cloud resource scaling. IEEE Network Operations and Management Symposium, 2012, 104(5): 1327-1334

[9] Calheiros R, Masoumi E, Ranjan R, Buyya R. Workload prediction using ARIMA model and its impact on cloud applications' QoS. IEEE Transactions on Cloud Computing, 2015, 3(4): 449-458

[10] Zhang S, Qian Z, Luo Z, et al. Burstiness-aware resource reservation for server consolidation in computing clouds. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(4): 964-977

[11] Richman J S, Moorman J R. Physiological time-series analysis using approximate entropy and sample entropy. AJP Heart and Circulatory Physiology, 2000, 278(6): 2039-2049

[12] Casale G, Mi N, Smirni E. Model-driven system capacity planning under workload burstiness. IEEE Transactions on Computers, 2009, 59(1): 66-80

[13] Jiang J, Lu J, Zhang G, Long G. Optimal cloud resource auto-scaling for web applications//Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. Delft, Netherlands, 2013: 58-65

[14] Ali-Eldin A, Kihl M, Tordsson J, Elmroth E. Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control//Proceedings of the ACM Workshop on Scientific Cloud Computing DATE. New York, USA, 2012: 31-40

[15] Zou B X, Liu Q. ARMA-based traffic prediction and overload detection of network. Journal of Computer Research and Development, 2002, 39(12): 1645-1652

[16] Roy N, Dubey A, Gokhale A. Efficient autoscaling in the cloud using predictive models for workload forecasting// Proceedings of the IEEE International Conference on Cloud Computing. Washington, USA, 2011: 500-507

[17] Tran V G, Debusschere V, Bacha S. Hourly server workload forecasting up to 168 hours ahead using seasonal ARIMA model//Proceedings of the IEEE International Conference on Industrial Technology. Athens, Greece, 2012: 1127-1131

[18] Li S, Wang Y, Qiu X, et al. A workload prediction-based multi-VM provisioning mechanism in cloud computing// Proceedings of the IFIP/IEEE, 15th Asia-Pacific Network Operations and Management Symposium. Hiroshima, Japan, 2013: 1-6

[19] Baldan F J, Ramirezgallego S, Bergmeir C, et al. A forecasting methodology for workload forecasting in cloud systems. IEEE Transactions on Cloud Computing, 2016, (99): 1-1

[20] Wei G, Ling Y, Guo B, et al. Prediction-based data aggregation in wireless sensor networks: Combining grey model and Kalman filter. Computer Communications, 2011, 34(6): 793-802

[21] Kayacan E, Ulutas B, Kaynak O. Grey system theory-based models in time series prediction. Expert Systems with Applications, 2010, 37(2): 1784-1789

[22] Jheng J J, Tseng F H, Chao H C, Chou L D. A novel VM workload prediction using grey forecasting model in cloud data center//Proceedings of the IEEE International Conference on Information Networking. Phuket, Thailand, 2014: 40-45

[23] Tseng F M, Yu H C, Tzeng G H. Applied hybrid grey model to forecast seasonal time series. Technological Forecasting and Social Change, 2001, 67(2-3): 291-302

[24] Islam S, Keung J, Lee K, Liu A. Empirical prediction models for adaptive resource provisioning in the cloud. Future Generation Computer Systems, 2012, 28(1): 155-162

[25] Imam M T, Miskhat S F, Rahman R M, Amin M A. Neural network and regression based processor load prediction for efficient scaling of grid and cloud resources //Proceedings of the International Conference on Computer and Information Technology. Dhaka, Bangladesh, 2011: 333-338

[26] Qiu F, Zhang B, Guo J. A deep learning approach for VM workload prediction in the cloud//Proceedings of the IEEE/ACM International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/distributed Computing. Shanghai, China, 2016: 319-324

[27] Messias V R, Estrella J C, Ehlers R, et al. Combining time series prediction models using genetic algorithm to autoscaling Web applications hosted in the cloud infrastructure. Neural Computing and Applications, 2016, 27(8): 2383-2406

[28] Ghelardoni L, Ghio A, Anguita D. Energy load forecasting using empirical mode decomposition and support vector regression. IEEE Transactions on Smart Grid, 2013, 4(1): 549-556

[29] Bankole A A, Ajila S A. Cloud client prediction models for cloud resource provisioning in a multitier web application environment//Proceedings of the IEEE International Symposium on Service Oriented System Engineering. Redwood City, USA, 2013: 156-161

[30] Grechanik M, Luo Q, Poshyvanyk D, Portere A. Enhancing rules for cloud resource provisioning via learned software performance models //Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering. New York, USA, 2016: 209-214

[31] Yang J, Yu T, Jian L R, et al. An extreme automation framework for scaling cloud applications. IBM Journal of Research and Development, 2011, 55(6): 410-421

[32] Sun J, Chen H, Yin Z. AERS: An autonomic and elastic resource scheduling framework for cloud applications// Proceedings of the IEEE International Conference on Services Computing. San Francisco, USA, 2016: 66-73

[33] Coutinho E F, Gomes D G, Souza J N D. An autonomic computing-based architecture for cloud computing elasticity// Proceedings of the Network Operations and Management Symposium. Joao Pessoa, Brazil, 2015: 111-112

[34] Wager S, Hastie T, Efron B. Confidence intervals for random forests: The jackknife and the infinitesimal jackknife. Journal of Machine Learning Research, 2014, 15(1): 1625-1651

[35] Olinsky A, Kristin K, Brayton K B. Assessing gradient boosting in the reduction of misclassification error in the prediction of success for actuarial majors. Csbigs Cases in Business Industry Government & Government Statistics, 2014, 5(1): 12-16

[36] Ali-Eldin A, Rezaie A, Mehta A, et al. How will your workload look like in 6 Years? analyzing Wikimedia's workload//Proceedings of the IEEE International Conference on Cloud Engineering. Boston, USA, 2014: 349-354

[37] Sladescu M, Fekete A, Lee K, Liu. A. A polymorphic model for event associated workload bursts//Proceedings of the IEEE Institute Conference on Distributed Computing Systems Workshops. Philadelphia, USA, 2013: 119-125

[38] Huang C J, Guan C T, Chen H M, et al. An adaptive resource management scheme in cloud computing. Engineering Applications of Artificial Intelligence, 2013, 26(1): 382-389

[39] Yin J, Lu X, Chen H, et al. System resource utilization analysis and prediction for cloud based applications under bursty workloads. Information Sciences, 2014, 279: 338-357

[40] Yin J, Lu X, Zhao X, et al. BURSE: A bursty and self-similar workload generator for cloud computing. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(3): 668-680

[41] Gusella R. Characterizing the variability of arrival processes with indexes of dispersion. IEEE Journal on Selected Areas in Communications, 2006, 9(2): 203-211

[42] Casale G, Mi N, Smirni E. Model-driven system capacity planning under workload burstiness. IEEE Transactions on Computers, 2009, 59(1): 66-80

[43] Meng Yu, Zhang Bin, Guo Jun. Interval forecasting model of concurrent volume of cloud service users in cloud computing environment. Chinese Journal of Computers, 2017, 40(2): 378-396(in Chinese)
(孟煜, 张斌, 郭军. 云计算环境下云服务用户并发量的区间预测模型. 计算机学报, 2017, 40(2): 378-396)

[44] Tai J, Zhang J, Li J, et al. ArA: Adaptive resource allocation for cloud computing environments under bursty workloads//Proceedings of the IEEE International Performance Computing and Communications Conference. Orlando, USA, 2011: 1-8

GUO Jun, born in 1974, Ph. D., associate professor. His research interests include service computing and cloud computing.

WU Jing, born in 1992, M. S. candidate. Her research interests include cloud computing, bursty workload process and service optimization.

XING Liu-Dong, born in 1975, Ph. D., professor, Ph. D. supervisor. Her research interests include cloud computing, reliability modeling and analysis of complex system.

ZHANG Bin, born in 1964, Ph. D., professor, Ph. D. supervisor. His research interests include service computing and cloud computing.

ZHANG Rong, born in 1992, M. S. Her research interests include cloud computing, service prediction and optimization.

Background

With the increase in numbers and types of cloud services, change rules of workload from cloud services are becoming more and more various and complicated. Bursty workload of low regularity and high surges widely exists in cloud services, which not only lengthens request response time, and increases request violation along with rejection rate, but also breaks the balance in resource utilization rate among different layers of multi-layer applications.

In order to ensure QoS(Quality of Service) under bursty workload, reactive resource allocation and proactive resource reservation are developed. What's more, the former usually predefines a series of triggering thresholds and adjustment actions, and only metrics of systems exceeding the thresholds can the actions be taken. However, this strategy holds too poor timeliness when handling long-lasting and huge-spikes bursty workload because it requires that system starts up a large amount of resources instantaneously.

Proactive resource reservation strategy relies on statistic models or prediction models to confirm the amount of resources to be reserved to guarantee QoS. Moreover, workload peaks are selected as foundations for statistic models to conclude the resources amount. But if the peaks have a considerable difference with means and also occur with a low frequency, system will suffer from a waste of resources. On the contrary, prediction models conduct forecast in resource demands, then according to prediction results to dynamically plan and execute adjustment actions before the arrivals of real workload, thus this strategy performs better in timeliness. Nevertheless, there are two disadvantages in predictive strategies as overly strict regulations in workloads' types and unacceptable under-estimate in resource demands, which decrease the accuracy of prediction results, and deteriorate the effectiveness of adjustment actions. Besides, current strategies keep coarse-grained and low-precision in evaluating available resources, which just adopt initial configurations of resources as available value.

This paper proposes a novel VM cache mechanism that efficiently avoids long-lasting congestions of bursty workloads to cache queues, and betters QoS. At the same time, the coping strategy given by this work takes the advantage of great timeliness of resource reservations based on prediction models. Additionally, it modifies prediction models to make them own lower under-estimate in resource demands, and improves evaluation algorithms in available resources by integrating resource usage conditions, SLA (Service Level Agreement) with initial configurations in order to enhance effectiveness of the coping strategy.

This work was supported by the National Natural Science Foundation of China (61300019, 61370155), and the Special Fund for Fundamental Research of Central Universities of Northeastern University (N120804001).