物联网云服务的动态流量控制策略研究

凌其能，王一川，杨贾冰，郑侃
（北京邮电大学信息与通信工程学院，北京 100876）

摘　要：在云服务系统中，流量控制对于保证服务质量非常重要，提出了一种动态两级流量控制策略。利用基于实际云服务系统的改进最小连接数（least connection）算法，计算服务节点的综合负载，对各个服务节点进行负载均衡，并合理分配负载。在服务节点内部，提出 Zookeeper 动态分级调整阈值的策略，对流量进行动态控制。实际云服务系统性能测试表明，采用动态两级流量控制策略，服务节点的 RPS 明显优于未采用该策略的 RPS，且服务节点的资源利用率有所提高。

关键词：负载均衡；动态调整；流量控制；云服务系统
中图分类号：TN915.07
文献标识码：A
doi: 10.11959/j.issn.2096-3750.2018.00082

Research on dynamic traffic control strategy of IoT cloud service

LING Qineng, WANG Yichuan, YANG Jiabing, ZHENG Kan
School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China

Abstract: Flow control is very important to ensure quality of service in cloud service system. A dynamic two-level flow control strategy was proposed. Using the improved minimum connection number algorithm based on the actual cloud service system, the integrated load of the service node was calculated, load balancing was performed on each service node and the load was distributed reasonably. Within the service node, a dynamic hierarchical threshold adjustment strategy was proposed to control traffic dynamically. The actual cloud service system performance test shows that with the dynamic two-level flow control strategy, the RPS of the service node is significantly better than the RPS without the strategy, and the resource utilization of the service node is improved.

Key words: load balancing, dynamic adjustment, flow control, cloud service system

1　引言

互联网作为新一代信息技术的载体，是信息化社会的重要表征[1-2]。而物联网（IoT, Internet of things）作为新一代信息技术的重要组成部分，也是信息化时代的重要发展阶段。物联网意味着所有设备都以一个独特的身份（如 IP）存在，使用此身份与其他设备进行连接和通信[3]。云服务是基于互联网相关服务的增加、使用和交互模式，模式主要分为 3 种：软件即服务（SaaS）、平台即服务（PaaS）和基础设施即服务（IaaS）[4]。

物联网云服务的提出在很大程度上满足了人们对网络的各种需求，这意味着其用户数将会很庞大，与此同时，从客户端发给云平台服务器的流量也将出现暴涨，云平台中的资源被急剧消耗。为了保证云服务的稳定性和可靠性，当网络资源出现瓶颈时，需要对网络流量进行控制，传统的流量控制一般采用配置预分配策略，在平台部署时，各个服务节点按照分配的静态阈值进行流量控制，超出流量控制阈值的请求则被拒绝访问。但是，当服务节点发生变化或某个服务节点异常时，静态预分配方案无法有效进行流量控制。而在实际平台系统中，需要处理大量的服务请求，

收稿日期：2018-10-30；**修回日期**：2018-11-15
基金项目：国家自然科学基金资助项目（No.61671089）
Foundation Item: The National Natural Science Foundation of China (No.61671089)

服务节点异常的情况经常出现，此时利用动态流量控制策略对网络流量进行有效控制，可以有效利用带宽，降低网络服务时延，提高云服务质量。

在流量控制策略的研究基础上，负载均衡算法得到广泛应用，常见的用于分布式系统中的负载均衡算法按照有、无动态反馈性分为以下两类。

1) 静态负载均衡算法：主要包括文献[5]中采用的轮询（round-robin）算法、文献[6-7]中介绍的加权轮询（weighted round-robin）算法和随机（random）算法等。

2) 动态负载均衡算法：主要包括文献[8]采用的 Least Connection 算法、文献[9-10]采用的加入空闲队列（join idle queue）算法等。

文献[8]中介绍了一种带标记的 Least Connection 算法，利用 Gebrehiwot M E, Aalto S 等人提出的 Power Provisioning Policy 对服务器进行标记，从而选择优先分配负载服务器。本文提出了一种基于实际云服务系统的动态两级流量控制策略，利用 Least Connection 算法对各个服务节点进行负载均衡；通过实际云服务平台的需求对算法的参数进行适应性调整；提出了改进的 Least Connection 算法，之后在各个服务节点内部添加限流策略并通过分布式协调系统 Zookeeper 动态更新限流模块配置，从而达到动态流量控制的目的。最后通过实际云服务系统测试表明，这种动态流量控制策略在实际云服务平台中是行之有效的。

2　物联网云服务系统简介

物联网的发展促进了信息社会的进步，也不断为生活带来便利，对于物联网云服务系统来说，主要关注 3 个方面：设备接入、数据管理和通信方式[11-13]。

物联网云服务系统示意图如图 1 所示，在设备接入方面，在实际物联网云服务系统中，浏览器、智能终端及一系列 IoT 设备都通过 TCP-Connection（TCP 长连接）的方式与系统中的服务器建立连接。在数据管理方面，设备上报的数据通过 MQTT 服务器进行上报，并将数据有效存储在 Redis、MongoDB 等数据库中。在通信协议的选择方面，采用超文本传输协议（HTTP）和消息队列遥测传输协议（MQTT），客户端通过向 HTTP 服务器发送请求，获得相应的数据消息，MQTT 服务器则保证客户端与其之间建立一个长连接，通过发布/订阅方式以及 MQTT 服务器中介，实现发送消息客户端与订阅消息客户端之间的数据通信。

在实际云服务系统中，云服务监控系统主要负责收集各个服务节点的性能数据，并通过 DeviceShadow[14]服务及时更新数据库存储的数据；另一方面，在监控系统中，设计了 Web 展示页面，将各个服务节点的性能数据直观地展示在网页上。通过网页的数据展示，可以检测服务节点的性能参数，例如 CPU（CPU 利用率）、MEM（内存使用率）、响应时间、请求总数等。此外，云服务监控系统提供了相应的接口供用户调用，从而获取监控的服务节点数据。

另外，使用 Zookeeper[15]作为配置中心进行限流相关配置信息的管理，如实际分布式云服务系统中的某个服务节点限流配置信息保存在 Zookeeper 的某个目录节点下，当服务节点因为性能变化的原

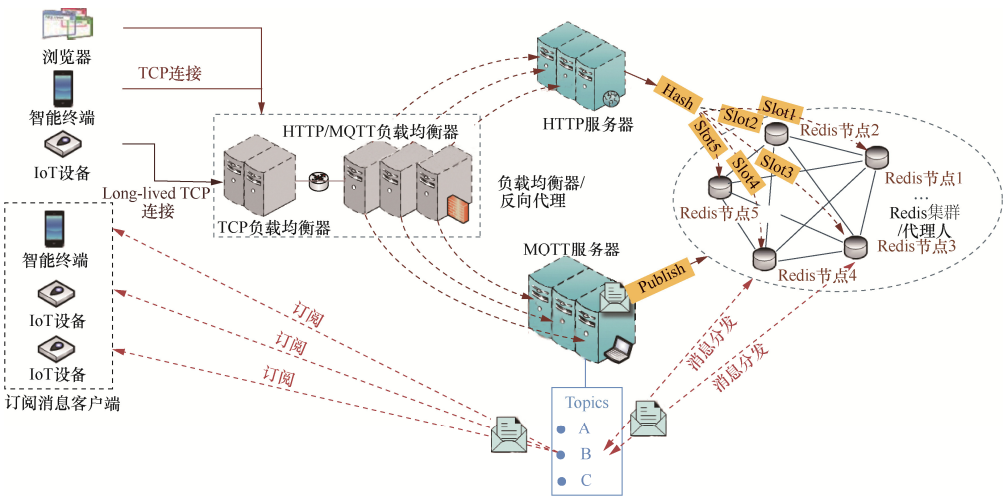


图 1　物联网云服务系统示意图

因需要更新限流的配置信息时，该服务节点会收到 Zookeeper 的通知，然后从 Zookeeper 获取新的配置信息并应用到服务节点内部。

3　动态流量控制策略

根据物联网云服务的分布式架构，本文设计的动态流量控制策略可以分为两级，由于服务节点存在不同的性能或需求，在第一级，采用 Least Connection 算法对各个服务节点进行负载均衡，将流量按照一定的规则分配给各个服务节点。在第二级，即各个服务节点内部，采用 Leaky Bucket 算法对服务节点内部的流量进行流量限制，当监控平台监控到各个服务节点的性能指标时，会根据限流级别动态调整流量阈值。由于在实际云服务系统中，各个服务节点的处理性能近似，所以采用 Least Connection 算法进行负载均衡，下面主要介绍 Least Connection 算法原理以及在实际云服务系统中的改进。

3.1　Least Connection 算法

3.1.1　基本介绍

Least Connection 算法是将网络连接指向建立连接数最少的服务器，由于需要动态计算各个服务器的连接数以估计其负载，所以被称为一种典型的动态负载均衡算法。此算法在内部需要对每个服务节点的连接数进行数据记录，当一个请求被调度到某台服务器时，其连接数加一；当连接超时或者结束中止，则连接数减一。

3.1.2　具体流程

为了保证服务节点的正常处理，在使用 Least Connection 算法时，会为每个服务节点引入一个额外的权值。如果实际云服务系统中，每个服务节点的额外权值出现为零的情况，则说明该服务器已经过载或者出现其他故障，此时不再给这个服务节点分配任务。

假设服务节点列表为

$$S=\{S_0,S_1,S_2,\cdots,S_{n-1}\}$$

定义 $W(S_i)$表示服务节点 i 的额外权值，$C(S_i)$表示当前服务节点的连接总数。具体算法描述如下。

algorithm 1　Least Connection algorithm
input: $W(S)$ Extra weight of server S
　$C(S)$ Total number of connections of server S
output: server with least connections
　for each server m in all servers do
　if $W(S_m)$ >0 then
　for server m+1 in all server n do
　if $W(S_i)$ <0 then
　Drop this server
　end if
　if $C(S_i)$ < (S_m) then
　m=1
　end if
　end for
　return server m
　end if
end for

简单来说，当服务器的额外权值大于 0 时，按照最小连接数原则为服务节点分配负载，即对所有服务节点的 $C(S_i)$进行遍历，选取最小连接数的服务节点。当额外权值小于或者等于 0 时，则跳过该服务节点，寻找下一个服务节点，直到找到当前连接数最小的服务节点。

从以上的算法描述可知 Least Connection 算法的基本思想，对服务节点进行任务调度时，满足连接数最小的服务节点优先进行任务处理。但是通过分析可知，Least Connection 算法只是把连接数近似看成服务节点的负载，并不能精确反映服务节点的负载情况。例如，当某个服务节点的实际负载增大、处理能力下降且连接数没有上升时，按照 Least Connection 算法，当再给这个服务节点分配任务时，其实际负载会越来越大，不利于服务节点的正常运行。为了保证服务节点能够高效、稳定地运行，基于实际云服务系统的需求，本文对 Least Connection 算法进行了一定程度的改进。

3.1.3　改进的 Least Connection 算法

对于实际云服务系统中的服务节点，其真实负载随请求数、资源利用率不断变化。基于此条件，改进的基本情况如下。根据之前提到的监控平台，定时（如周期 T 为 2 s）从监控平台收集各服务节点的性能参数。以 HTTP 服务为例，假设各个服务节点的性能近似，通过监控平台可以得到一些数据，包括周期内该服务节点占请求总数的几率 $L(R_i)$、服务节点的 CPU 使用率 $L(C_i)$和服务节点内存使用率 $L(M_i)$。

针对云服务实际系统，将各个参数的权值定义为 r_i，权值越高，则该参数越重要。根据上述 3 个参数，可以得到各个服务节点的综合负载，具体计算式如下

$$A=\begin{bmatrix}r_1 & r_2 & r_3\end{bmatrix}\begin{bmatrix}L(R_i)\\L(C_i)\\L(M_i)\end{bmatrix} \tag{1}$$

$$Load_i = \det(A) \tag{2}$$

其中，$\sum_{i=1}^{n} r_i = 1, i = 1,2,3,\cdots,n$

基于实际云服务系统的改进思想，即利用服务器 i 的综合负载 Load_i，对请求进行分配，而不仅通过 Least Connection 算法中的连接数估算负载，再对请求进行分配。目前，针对实际云平台系统，定义权值矩阵 r 为

$$r = \begin{bmatrix} 0.4 & 0.5 & 0.1 \end{bmatrix}$$

通过此权值，每周期 T 计算一次各个服务节点的综合负载 Load_i，当服务节点 i 满足

$$\text{Load}_i = \min(\text{Load}_1, \text{Load}_2, \cdots, \text{Load}_n)$$

则优先给服务节点 i 分配负载。

基于实际云服务系统中改进的负载均衡算法，客户端向服务节点发送请求时，先通过第一级的负载均衡模块，将请求优先分配给根据计算得到的综合负载最小的服务节点；然后在各个服务节点内部，进行第二级的流量控制。

3.2　流量控制算法

在本文动态流量控制策略中的第二级，采用漏桶（leaky bucket）[16]算法进行流量控制。漏桶算法是流量速率限制过程中的一种常见算法。该算法的核心思想是控制请求进入网络的速率，平缓网络上的突发流量。图 2 为漏桶算法基本原理。

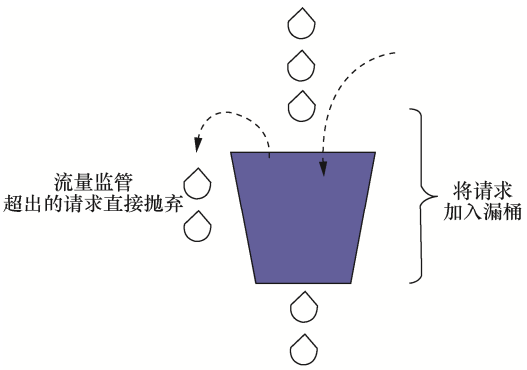


图 2　漏桶算法基本原理

漏桶算法可以看成由固定速率和有限容量的缓冲器组成的排队服务系统，其主要参数是 σ、C、r 和 T，其中 σ 代表缓冲器的大小，C 代表输出请求流的速率，单个请求的处理时间为 T，输入请求流速率为 r。假设漏桶已满时存在排队，定义 c 为此时队列中的请求数，w 为系统中需要排队的请求做出决策之后该队列中的请求数。则存在以下关系

$$C = 1/T \quad \begin{cases} r < \sigma \\ r > \sigma, w = 0 \\ r > \sigma, w = c \end{cases} \tag{3}$$

当某个请求到达某个服务节点时，首先检查当前该服务节点的漏桶状态。如果漏桶当前还有一定的余量，便可以将请求放入漏桶中，等待服务节点进行处理。如果漏桶已满，则服务节点存在两种选择：一种选择是放弃当前的请求，即式(3)中 w=0 的情况，该策略称为流量监管（traffic policing）；另一种选择是等待漏桶中之前的请求处理完，则漏桶有空间接受当前的请求，即式(3)中 w=c 的情况，该策略称为流量整形（traffic shaping）。在本文的实际云服务系统中，采用流量监管策略，即当请求数超出漏桶容量时，将超出漏桶容量的请求抛弃，从而达到流量控制的目的。

3.3　实现过程

上文提出的动态流量控制策略在实际云服务系统中的实现流程如图 3 所示。

在实际云服务系统中，动态流量控制策略主要由负载均衡模块、限流模块、监控平台和 Zookeeper 服务 4 个子模块协同实现。由于监控平台每个周期内会获取各个服务节点的性能参数，依照之前提出的权重，计算各个服务节点的综合负载，然后保存综合负载最小的服务节点。当网络请求到达服务系统时，负载均衡模块获得综合负载最小的服务节点，给其优先分配请求，该过程完成了动态流量控制策略的第一步。

服务节点收到相应请求后，在服务节点内部首先会通过限流模块，即上一节中提出的基于漏桶算法流量控制方案设计的限流模块，当超出限流设定阈值时，限流模块将会把之后的请求抛弃。而对于 Zookeeper 服务，当监测到某个服务节点触发了限流，便会通过监控平台检查该服务节点的 CPU，本文中暂定义 CPU 代表服务节点的健康度，若其 CPU 未达到相应的限制，则服务节点向 Zookeeper 服务更新服务节点的阈值以提高该服务节点的限流阈值。具体的限流策略流程如图 4 所示。

4　性能与分析

为了验证本文中动态流量控制策略的合理性，通过在实际云服务系统中进行多层次、不同条件下

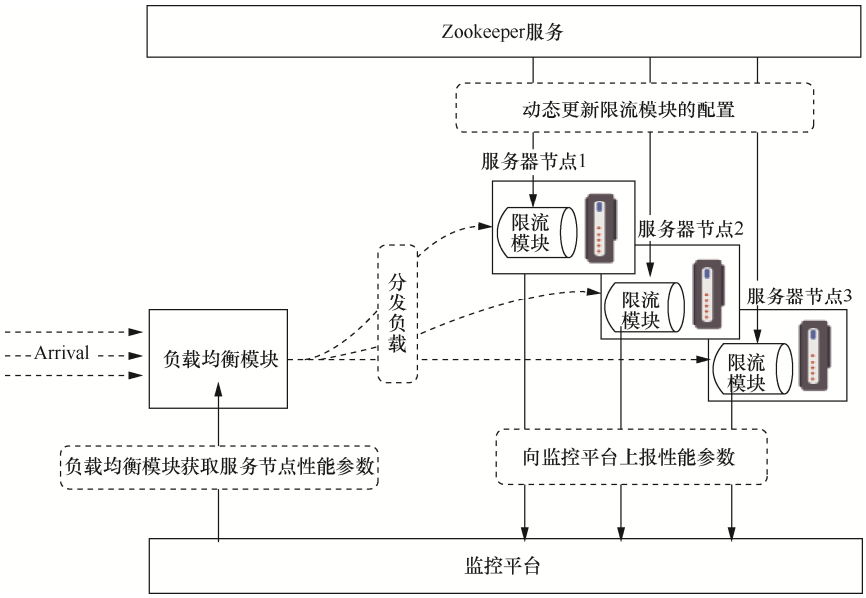


图 3　动态流量控制策略示意图

的测试，并与没有采用动态流量控制策略的情况进行对比，证明在实际云服务系统中，采用动态流量控制策略有助于提高服务节点的性能和资源利用率，同时 RPS 也有提高。

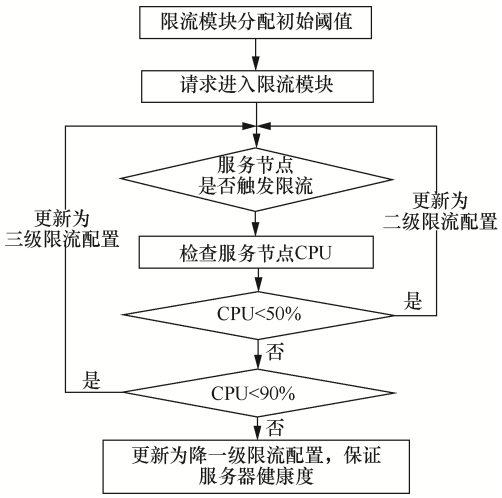


图 4　限流策略流程

4.1　测试环境

在实际云服务系统中，服务器配置如表 1 所示。

在服务器方面，本文针对 HTTP 服务进行测试，在服务器上运行了 4 个 HTTP 服务器；在客户端方面，针对低并发和高并发两种情况分别进行测试，具体模拟并发测试的工具为 Locust。Locust 是一个易于使用的、分布式的用户负载压力测试工具，在测试过程中，会产生大量的模拟客户端，每个模拟客户端按照脚本定期进行请求操作。Locust 旨在对服务节点的性能进行压力测试，研究在高并发和低并发两种压力条件下，服务节点的处理能力。

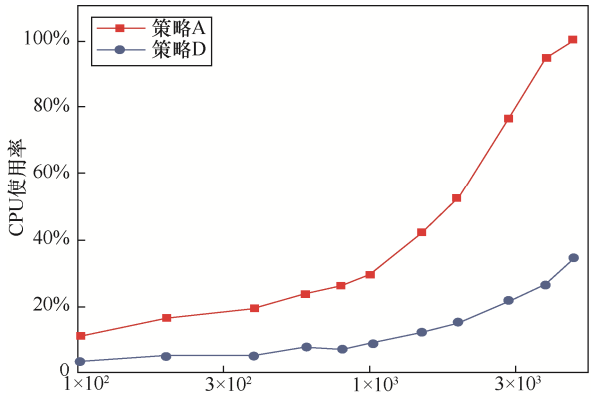
表 1　服务器配置

CPU	Intel(R) Xeon(R) CPU E5-2680
Clock speed	2.70 GHz
RAM	8 GB
Host	1
Core	5

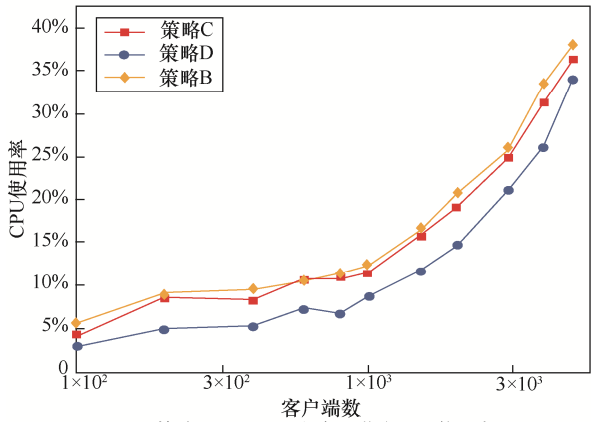
4.2　测试结果与分析

如图 5 所示，本文分别测试了在实际云服务系统中，未采用任何流量控制策略（策略 A）、采用 Least Connection 策略（策略 B）、采用 Round-Robin 并进行内部限流策略（策略 C）以及本文提出的改进 Least Connection 算法的动态流量控制策略（策略 D）对于服务节点处理性能的影响。如图 5(a)所示，当模拟客户端数量不断增加时，未采用任何策略的策略 A 在客户端数达到 5 000 个的情况下，CPU 使用率接近 100%，服务节点基本处于无法工作的状态。而本文提出的策略 D 在此情况下表现相对良好，服务节点的 CPU 使用率仅不到 40%。如图 5(b)所示，针对均采用负载均衡策略的情况，在 Round-Robin 算法基础上增加限流模块的策略 C 和仅使用基本 Least Connection 的策略 B 在实际系统中的 CPU 使用率均高于本文提出的策略 D。因此，在相同并发条

件下，动态流量控制策略对服务节点的 CPU 使用率具有良好的优化表现。



(a) 策略A、D下服务器节点CPU使用率



(b) 策略C、D、B下服务器节点CPU使用率

图 5　不同策略下服务节点 CPU 使用率

针对 RPS（每秒请求数），本文分别测试在实际云服务系统中，采用 Round-Robin、基于实际云服务系统的 Least Connection 算法和未采用任何策略 3 种条件下的最大 RPS 值。不同策略下服务节点最大 RPS 如图 6 所示，对于 RPS 的极值，采用基于实际云服务系统的 Least Connection 算法，服务节点的 RPS 值最大，且相对于未采用任何策略的系统，RPS 值提升接近 50%。

可见，通过本文提出的动态两级流量控制策略，在实际云服务系统中服务节点的资源使用率得到提高，处理请求的性能也有所提升，对于保证物联网云服务系统的网络质量是有效的。

5　结束语

本文研究了基于实际云服务系统的流量控制策略，并提出了一种动态两级流量控制策略。通过负载均衡，在负载分发之前，找到综合负载最小的服务节点优先进行负载分发。之后，在服务节点内部通过限流策略进行流量控制，并且通过 Zookeeper 动态调整限流策略的阈值。经实际云服务系统中的测试表明，提出的动态流量控制策略对于提升云服务系统中服务节点的处理性能有效。在大规模流量控制方面，为了保证实际的网络服务质量，还需要在网络请求的行为和特征上进行更加深入的探究。

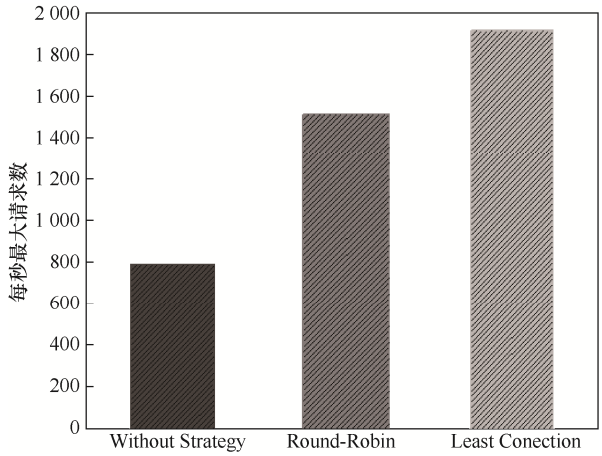


图 6　不同策略下服务节点最大 RPS

参考文献：

[1] ZHENG K, LIU F, LEI L, et al. Stochastic performance analysis of a wireless finitestate Markov channel[J]. IEEE Transactions on Wireless Communications, 2013, 12(2):782-793.

[2] ZHENG K, YANG Z, ZHANG K, et al. Big data-driven optimization for mobile networks toward 5G[J]. IEEE Network, 2016, 30(1):44-51.

[3] SYAL A S, GUPTA A. Internet of things: review on security of novel technology[C]//2017 International Conference on Smart Technologies For Smart Nation (SmartTechCon), August 17-19, 2017, Bangalore, India. Piscataway: IEEE Press, 2017:1405-1410.

[4] LEE B, TIM G, ROBERT C P, et al. DRAF-T cloud computing synopsis and recommendations[J]. Recommendations of the National Institute of Standards and Technology, 2011, (800): 46.

[5] SAMAL P, MISHRA P. Analysis of variants in Round Robin algorithms for load balancing in cloud computing[J]. International Journal of Computer Science and Information Technologies, 2013, 4(3):416-419.

[6] KASHYAP D, VIRADIYA J. A survey of various load balancing algorithms in cloud computing[J]. International Journal of Scientific & Technology Research, 2014, 3(11):115-119.

[7] AL NUAIMI K, MOHAMED N, AL NUAI-MI M, et al. A survey of load balancing in cloud computing: challenges and algorithms[C]//Network Cloud Computing and Applications (NCCA), December 3-4, 2012, London, UK. Piscataway: IEEE Press, 2012: 137-142.

[8] GEBREHIWOT M E, AALTO S, LASSILA P. Energy efficient load balancing in web server clusters[C]//2017 29th International Teletraffic Congress(ITC 29), September 4-8, 2017, Genoa, Italy. Piscataway: IEEE Press, 2017(3):13-18.

[9] LU Y, XIE Q, KLIOT G, et al. Join-Idle-Qu-eue: a novel load balancing algorithm for dynamically scalable web services[J]. Performance Evaluation, 2011, 68(11):1056-1071.

[10] CHOUDHARY M, CHANDRA D, GUPTA D. Load balancing algorithm using JIQ methodology for virtual machines[C]//2017 International Conference on Computing, Communication and Automation(ICCCA), May 5-6, 2017, Greater Noida, India. Piscataway: IEEE Press, 2017:730-735.

[11] KIM W, KIM S D, LEE E, et al. Adoption issues for cloud computing[C]//Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia, December 14-16, 2009, Kuala Lumpur, Malaysia. New York: ACM, 2009:2-5.

[12] HOU L, ZHAO S, XIONG X, et al. Internet of things cloud: architecture and implementation[J]. IEEE Communications Magazine, 2016, 54(12):32-39.

[13] ZHENG K, MENG H, CHATZIMISIOS P, et al. An SMDP-based resource allocation in vehicular cloud computing systems[J]. IEEE Transactions on Industrial Electronics, 2015, 62(12):7920-7928.

[14] TÄRNEBERG W, CHANDRASEKARAN V, HUMPHREY M. Experiences creating a framework for smart traffic control using AWS IoT[C]//Proceedings of the 9th International Conference on Utility and Cloud Computing, December 6-9, 2016, Shanghai, China. New York: ACM, 2016:63-69.

[15] HALOI S. Apache zookeeper essentials[M]. Bermingham: Packt Publishing Limited, 2015.

[16] YIN N, HLUCHYJ M G. Analysis of the leaky bucket algorithm for on-off data sources[J]. Journal of High Speed Networks, 1993, 2(1): 81-98.

[作者简介]

凌其能（1994-），男，北京邮电大学硕士生，主要研究方向为物联网、智能云计算等。

杨贾冰（1994-），男，北京邮电大学硕士生，主要研究方向为物联网、智能云计算等。

王一川（1994-），男，北京邮电大学硕士生，主要研究方向为物联网、智能云计算等。

郑侃（1974-），男，北京邮电大学教授、博士生导师，IEEE 高级会员、IET 会士，主要研究方向为车联网、无线网络安全、智能计算等。