基于软件定义网络的服务器集群负载均衡技术研究

于天放* 芮兰兰 邱雪松

(北京邮电大学 网络与交换技术国家重点实验室 北京 100876)

摘 要：在当前的网络体系结构下，采用硬件系统实现服务器集群负载均衡存在着获取负载节点状态困难、流量导向方式复杂等制约因素，不利于提升服务器集群的伸缩性和服务性能。针对此问题，该文提出一种基于软件定义网络(SDN)的负载均衡机制(SDNLB)。该机制借助SDN具有的集中式控制和流量灵活调度优势，利用SNMP协议和OpenFlow协议对服务器的运行状态和全局网络负载信息进行实时监测，并通过权值计算的方式选择出权重最高的服务器作为流处理的目标服务器，在此基础上，采用最优转发路径算法进行流量调度，从而达到提高服务器集群的利用率与处理性能的目的。搭建了实验平台对SDNLB的性能进行仿真测试，实验结果表明：在相同的网络负载条件下，SDNLB与其他负载均衡算法相比，能够有效地降低服务器集群的负载，并能够显著提高网络吞吐量和带宽利用率，缩短流的完成时间和平均时延。

关键词：软件定义网络；OpenFlow；服务器集群；负载均衡；流量工程

中图分类号：TP393 **文献标识码**：A **文章编号**：1009-5896(2018)12-3028-08

DOI: 10.11999/JEIT180207

Research on SDN-based Load Balancing Technology of Server Cluster

YU Tianfang RUI Lanlan QIU Xuesong

(*State Key Laboratory of Networking and Switching Technology*, *Beijing University of Posts and Telecommunications*, *Beijing* 100876, *China*)

Abstract: Under the present network architecture, it is disadvantageous for scalability and service performance of server cluster to adopt hardware systems to realize load balancing of server cluster, because there are some restriction factors in such a method, including the difficulty of acquiring load nodes status and the complexity of redirecting traffic, etc. To solve the problem, a Load Balancing mechanism based on Software-Defined Networking (SDNLB) is proposed. With superiorities of SDN such as centralized control and flexible traffic scheduling, SDNLB monitors run states of servers and overall network load information by means of SNMP protocol and OpenFlow protocol in real time, and chooses the highest weight server as target server aiming for processing coming flows through the way of weight value calculation. On this basis, SDNLB takes full advantage of the optimal forwarding path algorithm to carry on traffic scheduling, and achieves the goal that raises utilization rate and processing performance of server cluster. An experiment platform is built to carry out simulation tests for overall performance of SDNLB, and the experiment results show that under the same network load conditions, SDNLB lowers effectively loads of server cluster, noticeably raises network throughput and bandwidth utilization, and reduces finish time and average latency of flows, compared with other load balancing algorithms.

Key words: Software-Defined Networking (SDN); OpenFlow; Server cluster; Load balancing; Traffic engineering

1 引言

随着云计算、大数据等新兴技术的不断涌现，数据中心等大型机构通常采用多台服务器组成集群的方式来负载多用户的接入和任务请求，并借助专用的负载均衡器将系统负载分配到不同的节点进行处理，避免服务器出现单点故障造成流量的损失。负载均衡器所采用的调度算法是影响负载均衡效果的关键因素。一般而言，负载均衡算法主要分为无状态算法和有状态算法。较典型的无状态算法包括轮询调度算法RR[1]、目标地址散列算法DH[2]等。这些算法具有实现简单、易于部署的特点，在执行

收稿日期：2018-02-28；改回日期：2018-08-13；网络出版：2018-08-22
*通信作者： 于天放 m2015010@foxmail.com
基金项目：国家自然科学基金(61702048, 61302078)
Foundation Items: The National Natural Science Foundation of China (61702048, 61302078)

过程中不依赖于任务的性质、执行时间和服务器状态的变化，因此会出现服务器之间的负载无法随着网络变化动态进行调整的情况，从而造成服务器集群整体性能的下降。相比无状态算法，有状态算法则更加灵活，通过实时获取网络的负载状况，动态地将任务请求分发到服务器中，减少了集群负载的不平衡性。典型的有状态算法包括最少连接算法、最快响应速度算法等。由于有状态算法采用连接数、任务数等指标来表示负载，当服务器集群提供多种服务时，相同的连接数或任务数并不能代表相同的负载量，因此，每种服务请求带来的负载量就会各不相同，不能真实反映服务器节点的负载状况。

鉴于上述问题，文献[3]提出一种改进的加权最少连接算法，来提高多层次服务器集群架构的伸缩性和负载性能。文献[4]针对存储集群系统中热点数据节点访问不均衡问题，提出一种SARM机制，该机制通过在服务器节点创建数据副本，并根据节点的负载状况自动调用不同的调度算法以实现对用户请求进行合理调度，减少通信开销。文献[5]对自适应机制在解决服务器集群过载问题存在的不足之处进行了分析，采用Viterbi算法构建了一种能够预测服务节点下一步状态的架构，根据预测结果，该架构会选择基于不同决策参数的自适应策略进行流量负载均衡操作，从而减少服务器运行时的不确定性和任务请求的响应时间。研究表明，尽管文献[3–5]提出的方法能够提高服务器集群的可靠性和服务能力，但仍然存在一些问题。首先，网络拓扑的高连通性没有得到充分利用，在面对瞬间高并发连接的突发流量时，这些方法没有达到理想的负载均衡效果。例如，文献[3]提出的负载均衡方法在计算路径时没有考虑当前链路的负载情况。其次，以上方法仅掌握局部流量分布信息，缺乏全局网络状态的路由与调度，使得网络吞吐量低于最优值。

SDN技术所具有的控制转发解耦合、网络可编程等特性为服务器集群负载均衡研究提供了崭新的思路，主要体现在：(1)SDN更容易获取整体网络拓扑结构，能够制定全局的路由策略，引导流量始终处于最优路径上。(2)SDN能够集中控制交换设备，通过控制器将转发策略以流表的形式统一下发到交换机中，而不用逐一配置每台交换机。目前，许多研究人员已开展了SDN环境下的流量负载均衡研究工作。Mao等人[6]设计了一种动态流表机制，该机制采用单表和组表相结合的方式对主机进行分类和流量监测，具有较好的适用性，能够提高网络流量调度的效率。文献[7]针对数据中心网络中大象流的路径配置问题，提出了一种2阶段负载均衡框架OFLoad。该框架对大象流工作负荷优化的同时，利用组表功能对网络中的小流进行聚合，采用倍增加权多路径路由算法进行路由分配。实验结果表明，在网络严重过载情形下，OFLoad通过对瓶颈路径赋予低权值，实现大象流的吞吐量不受到影响。文献[8]针对当前多路径路由算法存在的不足之处，提出了基于扩展OpenFlow协议的快速流调度策略Nimble。该策略通过监测交换机端口队列长度进行拥塞检测，同时依据数据包哈希值对大流进行标记，并采用GFF算法[9]对大流进行重路由。文献[10]面向树形数据中心网络结构提出了一种混合路由机制SHR，该机制根据预设的流分类阈值对数据流进行分类。对于小流，SHR采用随机选择上行路径的流量无视路由算法进行调度；对于大流，SHR采用基于队列长度的自适应路由算法进行调度。文献[11]提出了一种基于多路广播树的SDN多路径路由算法。该算法为拓扑中每个节点都创建一棵以该节点为根节点的无环广播树，并根据源、目的节点间各条路径的可用带宽和时延进行概率分配转发路径，仿真数据表明该算法能够有效地提高网络吞吐量，减少传输时延并降低控制器的负载率。

综上所述，本文在前人研究的基础上提出一种基于动态反馈的服务器集群负载均衡机制SDNLB。该机制作为控制器的一个模块，借助SDN技术集中式控制和流量灵活调度的优势，通过周期性地采集服务器运行状态和全局网络负载信息，利用最优转发路径算法进行流量调度，进而提高服务器集群的处理性能。具体来说，本文的主要工作如下：

(1) 设计了一种基于OpenFlow协议和LLDP[12](Link Layer Discovery Protocol)协议的拓扑发现方法，使得控制器能够根据网络中链路状态变化对全局网络拓扑视图进行维护。

(2) 设计了服务器性能监测方法。利用该方法控制器能够对服务器的运行状态进行分类，并通过权值计算选择出权重最高的目标服务器。

(3) 提出了一种最优转发路径算法，可以实现流量在多条候选路径中选择一条带宽利用率最低且长度最短的路径进行转发。

(4) 在SDN实验环境中实现了本文提出的负载均衡机制，并加以验证。

本文其他部分结构如下：第2节分析并构建了SDNLB评估模型，第3节对SDNLB的关键机制进行了阐述，第4节通过实验对SDNLB的性能进行验证，第5节总结全文。

2 SDNLB评估模型

2.1 评估指标1：目标服务器

服务器的瓶颈部件包括CPU、内存和磁盘，通过对这些部件的使用情况进行监测能够进一步判断服务器的运行状态。表1描述了服务器主要性能指标的具体分类，当一台服务器的某项指标处于过高状态时，说明该服务器已处于过载状态，不适合再处理新的连接请求。因此，在选择目标服务器的过程中，首先需要确定服务器的运行状态。SDNLB通过创建两个列表对服务器进行分类，分别为平稳状态服务器列表和过载状态服务器列表，列表的格式表示为：{服务器IP，系统配置，性能指标值}。SDNLB能够根据服务器的性能变化对这两个列表进行动态地调整，并对平稳状态列表中的服务器进行权值计算，选择权重最高的服务器作为目标服务器。

表 1 服务器主要性能指标

指标类型	指标值	状态
CPU利用率	非空闲任务占用比小于70%	良好
	70%~85%	过高
	90%以上	很差
内存访问	没有页交换	良好
	每个CPU每秒10个页交换	过高
	更多的页交换	很差
磁盘I/O	活动时间百分比小于30%	良好
	30%~45%	过高
	50%以上	很差

考虑到集群中服务器的系统配置各不相同，本文选择上述部件的固有参数和实时状态参数进行服务器的权值计算。

定义 1 固有参数：假定平稳状态列表中服务器数量为n，第i台服务器的CPU主频与核心数的乘积用C_i表示，物理内存大小用M_i表示，磁盘空间用D_i表示。为了准确描述不同参数的作用，对这些参数进行离差标准化处理，将原始数据转换为无量纲的纯数值。以参数C_i为例，其标准化公式可表示为

$$C_i' = \frac{C_i - \mathrm{Min}\,(C_i)}{\mathrm{Max}\,(C_i) - \mathrm{Min}\,(C_i)} \tag{1}$$

其中，C_i'为标准化后的数值，其范围为[0, 1]。相似地，参数M_i和参数D_i对应的标准化数值可分别表示为M_i'和D_i'。

定义 2 实时状态参数：假定第i台服务器的CPU利用率用U_i表示，内存占用率用H_i表示，磁盘使用率用S_i表示。根据定义1中的固有参数，对每个固有参数赋予相应的影响因子，采用加权平均方法确定服务器的总体性能P_i，即P_i表示为

$$P_i = \alpha C_i' + \beta M_i' + \gamma D_i' \tag{2}$$

其中，$\{\alpha, \beta, \gamma | \alpha, \beta, \gamma \in (0,1), \alpha+\beta+\gamma = 1\}$。相应地，第$i$台服务器的实时负载量$L_i$可表示为

$$L_i = \alpha C_i' U_i + \beta M_i' H_i + \gamma D_i' S_i \tag{3}$$

根据式(2)和式(3)，利用服务器的总体性能P_i减去实时负载L_i，即为服务器剩余的处理性能。每个服务器的剩余处理性能与列表中所有服务器的剩余处理性能之和的比值W_i就是该服务器的权重值。W_i可表示为

$$W_i = \frac{P_i - L_i}{\sum_{i=1}^{n}(P_i - L_i)} \tag{4}$$

因此，服务器权重序列$W = \{w_i | i = 1, 2, \cdots, n\}$，SDNLB将从$W$中选出权重最高的服务器作为目标服务器。

2.2 评估指标2：最优转发路径

SDNLB第2个评估指标是在网络中所有以目标服务器为目的节点的可达路径中选择满足约束条件的一条最优转发路径。本文采用图论构建相应的网络模型。若有向图G=(V, E)表示一个网络，其中V为OpenFlow交换机集合，E为网络中链路集合。$S(u, v)$表示从交换机u到交换机v的第1跳交换机集合。对于给定的源节点s和目的节点d，连通s和d的路径表示为$< s, u_0, \cdots, u_k, d >$，$u_k \in V$。当$j = 1, 2, \cdots, k$时，存在$(u_{j-1}, u_j) \in E$，且$u_j = S(u_{j-1}, d)$，则路径$< u_0, \cdots, u_k >$称为$s$到$d$的一条可达路径，$s$到$d$所有可达路径集合记为$\Phi_{sd}$。

定义 3 链路带宽利用率：对于任意一条链路$e \in E$, B_e表示链路e的容量，R_e表示链路e剩余的带宽，当接入带宽请求为δ的任务后，链路e的带宽利用率U_e表示为

$$U_e = \left(1 - \frac{R_e - \delta}{B_e}\right) \times 100\% \tag{5}$$

定义 4 路径可用带宽：对于任意一条路径$p \in \Phi_{sd}$, p的可用带宽A_p表示为该路径上所有链路的剩余带宽最小值，即

$$A_p = \min_{e \in p}(R_e) \tag{6}$$

定义 5 路径带宽利用率：对于任意一条路径$p \in \Phi_{sd}$, p的带宽利用率U_p表示为该路径上所有链路带宽利用率的最大值，即

$$U_p = \max_{e \in p}(U_e) \tag{7}$$

由于网络中的流量分布是不均匀的，为了减少

网络拥塞的发生，在路由过程中应将所选路径长度限制在一定范围内，并最少使用负载最重的链路来保证服务质量。因此，本文采用最小化最大带宽利用率(Max-Min)[13]原则进行最优路径选择，即首先在可达路径集合Φ_{sd}中选取满足$A_p > \delta$条件的前K条最短路径，并在这K条路径中选择一条路径p，保证U_p值最小，这样的路径p即为所求最优转发路径。如果存在多条这样的路径，则选择路径长度最少的一条作为转发路径。

SDNLB架构实现的主要功能组件包括以下几个方面：

(1) 拓扑发现组件。该组件利用OpenFlow协议和LLDP协议对网络中交换机之间的链路状态进行探测，并生成对应的全局网络拓扑视图。

(2) 性能监测组件。该组件通过获取服务器瓶颈部件的性能数据，动态维护平稳状态服务器列表和过载状态服务器列表，并通过权值计算的方式确定目标服务器。

(3) 链路状态采集组件。该组件通过获取交换机中流表计数器维护的端口传输数据信息，计算出网络中链路的可用带宽以及带宽利用率。

(4) 路由组件。该组件依据获得的全局网络拓扑视图和完整的链路负载信息，采用最优转发路径算法进行流量调度。

3　SDNLB关键机制与算法

本节对SDNLB的拓扑发现、服务器性能监测、链路状态采集等机制进行详细的描述，并给出具体的最优转发路径算法。

3.1　拓扑发现

在初始阶段，每个交换机向控制器发送特征应答消息告知自身的功能特性，消息中包含了交换机的数据路径识别号、所支持的流表数、报文缓冲区以及可用端口等信息。拓扑发现组件为每个交换机的所有可用端口都发送一个packet_out消息，该消息中携带了一个LLDP数据包，其数据单元由Chassis ID和Port ID构成。交换机收到消息后按照端口号分别转发给相邻的设备，如果邻居设备为交换机，则将消息中的数据包连同本身的元数据一起封装到packet_in消息中发送给控制器，拓扑发现组件对收到的消息进行解析，得到如下转发信息：(1)LLDP数据单元：{Chassis ID=源交换机id, Port ID=发送端口号}；(2)元数据：{Chassis ID=目的交换机id, Port ID=接收端口号}。根据这些信息，拓扑发现组件能够确定某两台交换机之间存在着链路关系，推广到整个网络，控制器就可以精确掌握全局的拓扑视图。由上述过程可知，若网络由m个交换机构成，每个交换机的可用端口数为n，交换机之间的连接数为l，拓扑发现组件需要发送和接收的消息数量分别用M_s和M_r表示，则

$$M_s = \sum_{i=1}^{m} n_i \tag{8}$$

$$M_r = 2l \tag{9}$$

在网络运行阶段，当某个交换机端口状态发生变化时，该交换机则向控制器发送端口状态消息，消息中指明了端口变化的原因。拓扑发现组件根据该消息内容再次进行拓扑信息的更新操作。

3.2　服务器性能监测

如图1所示，性能监测组件采用SNMP协议获取服务器瓶颈部件性能数据。通过在服务器中配置SNMP Agent服务，控制器能够对服务器的运行状态进行监测，动态地维护平稳状态服务器列表和过载状态服务器列表，进而确定目标服务器。

首先，性能监测组件通过GetBulkRequest原语创建请求报文，封装到IP数据包中发送给服务器。服务器中的SNMP代理进程负责对接收到的数据包进行解析，并按照请求报文中部件标识对服务器自身的管理信息库文件进行查询，最后通过Response原语将获得的部件性能数据封装到IP数据包中返回给控制器。性能监测组件对收到的数据包中的性能数据进行提取并以RRD(Round Robin

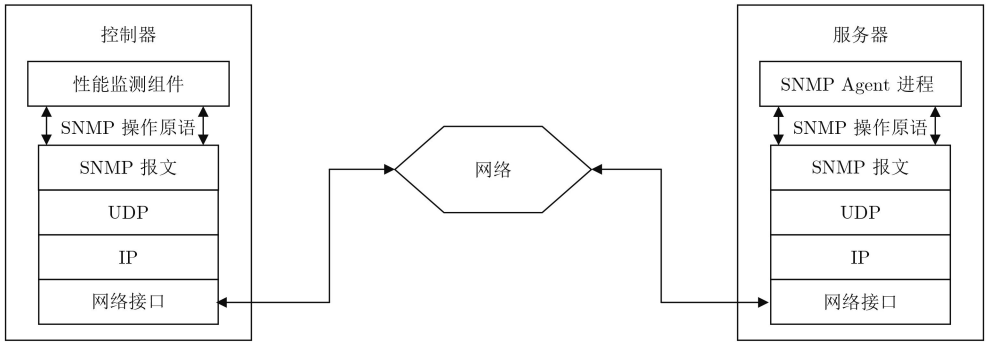


图 1 服务器性能监测的主要过程

Database)文件形式进行存储，保证数据的时序一致性和有效性。依照RRD文件中保存的数据信息，性能监测组件对服务器的当前运行状态标识为平稳状态或过载状态，同时对两个服务器状态列表进行更新操作。

3.3 链路状态采集

链路状态采集组件通过获取交换机中流表计数器维护的端口传输数据信息完成链路可用带宽及带宽利用率的测量。对于某条链路，控制器定期向该链路两端的交换机发送端口状态请求消息，交换机收到请求消息后返回应答消息，应答消息中包含了对应端口发送的字节数和接收的字节数等统计信息。假定在∇t时间内发送端交换机的连接端口发送的总字节数为X，接收端交换机的连接端口接收的总字节数为Y，令链路容量为B，则在该传输方向上的链路可用带宽R为

$$R = B - \mathrm{Min}\left(\frac{X}{\nabla t}, \frac{Y}{\nabla t}\right) \tag{10}$$

根据式(5)和式(10)就能计算得出该链路在∇t时间内的带宽利用率。

3.4 最优转发路径算法

当流进入网络中，路由组件依据目标服务器位置、当前的网络拓扑视图和网络链路负载信息计算所有能够接入该流的可达路径，并利用基于偏离路径的Yen算法[14]选择前K条最短路径。通过计算这K条路径的带宽利用率，选择带宽利用率最小且长度最少的一条路径作为流的转发路径。算法的具体描述如表2所示。

算法1的时间复杂度与选择前K条最短路径的时间和确定最优转发路径的时间密切相关。若图的顶点数为V，边数为E，采用斐波那契堆优化的Dijkstra算法求解源、目的节点间的最短路径的时间开销为$O(E+V\lg V)$。在选择前K条最短路径的过程中，需要调用KL次Dijkstra算法进行偏离路径的计算，L为偏离路径的长度，在最坏的情况下，L等于V，此过程的时间开销为$O(KV(E+V\lg V))$。在确定最优转发路径阶段，所有最短路径的带宽利用率最多需要进行K–1次比较，此过程的时间开销为$O(K)$。因此，算法1的时间复杂度为$O(KV(E+V\lg V)+K)$，即$O(KV(E+V\lg V))$。

表 2 流转发过程

算法1 SDNLB最优转发路径算法

输入： Topology_View /*当前网络拓扑视图*/
Link_Load /*网络链路负载信息*/
Target-Server /*目标服务器*/
Flow /*新到达的流*/

输出： R /*最优转发路径*/

(1) implement logical loopless processing
(2) destination=target-server
(3) create a graph that meets bandwidth demand of flow
(4) A=[ ]
(5) A[0]= Dijkstra(graph, source, destination)
(6) B=[ ]
(7) **for** i: 1 to K **do**
(8) **for** j: 0 to size($A[i-1]$)–1 **do**
(9) spur=$A[i-1]$.node(j)
(10) root=A[i –1].nodes(0, j)
(11) **for** path in A **do**
(12) **if** root==path.nodes(0, j) **do**
(13) remove path.nodes(j, j+1)
(14) **end if**
(15) **end for**
(16) spurpath=Dijkstra(graph, spur, destination)
(17) entirepath=root+spurpath
(18) **if** entirepath not in B **do**
(19) B.add(entirepath)
(20) **end if**
(21) recover those removed edges
(22) **end for**
(23) **if** B.length==0 **do**
(24) break
(25) **else do**
(26) B.sort()
(27) $A[i]$=B[0]
(28) B.delete(B[0])
(29) **end if**
(30) **end for**
(31) **if** A.length==1 **do**
(32) R=A[0]
(33) **else do**
(34) determine R by choosing a path from list A.Bandwidth utilization of the path should be minimum
(35) **end if**
(36) return R

4 性能评估

本节通过仿真实验对SDNLB的性能进行验证，并与ECMP(Equal Cost Multi-Path)算法、GFF算法和E-Dijkstra(Extending Dijkstra)算法[15]的性能进行定量的比较和分析。

4.1 仿真环境搭建

实验采用Mininet和Ryu搭建一个小型SDN网

络。Mininet可以快速部署到硬件环境中，其内置的OvS(Open vSwitch)虚拟交换机能够模拟出包含多终端和网络设备的复杂计算机网络。Ryu作为开源控制器允许用户添加模块实现自定义功能。实验拓扑采用Pod为4的Fattree结构，由1个控制器，20个OpenFlow交换机，16台终端主机构成。其中，终端主机的编号设置为1~16，编号为偶数的主机作为服务器集群运行，其余的主机作为客户机运行。网络拓扑中每条链路带宽均设置为10 Mbps，链路时延为2 ms。

4.2 仿真分析

ECMP算法的主要思想是将源、目的节点间所有的等价路径进行编号，并对分组头部信息进行哈希运算，通过将得到的哈希值与路径编号相匹配，从而确定一条转发路径。GFF算法是基于ECMP算法改进而来，当网络中有大流产生时，它将为该流线性搜索所有可能的有效路径，一旦发现满足其带宽需求的路径即进行转发。E-Dijkstra算法对Dijkstra算法进行了扩展，它分别将链路传输时延与交换机处理时延作为有向图中边的权重和节点的权重，以迭代的方式找到源、目的节点间一条时延最短的转发路径。

对于具有N个Pod的Fattree拓扑结构，处于同一Pod的相同边缘交换机上的各主机间的流量，只需访问到该交换机即可，主机间的路径唯一；处于同一Pod的不同边缘交换机上的各主机间的流量，最高需要访问到汇聚层交换机，主机间的路径数为$N/2$；对于不同Pod间的流量，首先确定源主机到核心层交换机的上行路径，相应地，核心层交换机到目的主机的下行路径也能够随之确定，因此任意两个主机间最多存在$(N/2)^2$条不同的路径。根据上述分析，实验通过设置服务器性能影响因子实现网络拓扑的流量分布，并在不同的网络负载条件下通过客户机发送长流和背景流两种类型的流量，其中长流的数据包大小为1470 Byte，每个流的发送时长为60 s；背景流的数据包大小为256 Byte，每个流的发送次数为15次。由于控制器需要定期采集服务器和链路的负载状态等信息，因此将其轮询周期设置为1 s。为了更准确地描述不同算法的性能差异，将网络负载以1 Mbps的迭代步长从1 Mbps增加到8 Mbps进行网络性能测试，并选择CPU组件作为衡量服务器负载状况的主要指标，根据定义2，这里将服务器性能影响因子设置为(0.5, 0.3, 0.2)。

表3和表4分别展示了不同算法下的CPU平均利用率和服务器的平均负载。由表中数据可以看出，在相同的网络负载条件下，相比于其他3种算法，SDNLB能够有效地减少CPU的计算开销，并降低服务器的负载，这反映出4种算法的执行效率是依次递减的，其原因在于E-Dijkstra算法在计算权值时没有考虑流速的影响，当网络中存在多种类型的流量时，并不能保证所选路径是最优的，而GFF算法在进行长流重路由的过程中，只要找到能满足流带宽需求的路径就进行转发，忽略了最优路径问题，同样地，ECMP算法采取了逐跳转发策略，假定所有路径的代价均相同，没有考虑链路实际负载等问题。

图2(a)展示了平均吞吐率[16]与网络负载之间的关系。随着负载量的不断增大，不同算法下的平均吞吐率也随之增长，在负载量为8 Mbps处，每种算法的平均吞吐率均达到最高值，证明了在一定范围内，网络的吞吐率与负载量是成正比关系。图2(b)展示了网络平均丢包率情况。从图中可看出，SDNLB的丢包程度最轻，且变化幅度较其他算法最小。在负载量为6 Mbps处，每种算法的丢包率

表 3　CPU平均利用率(%)

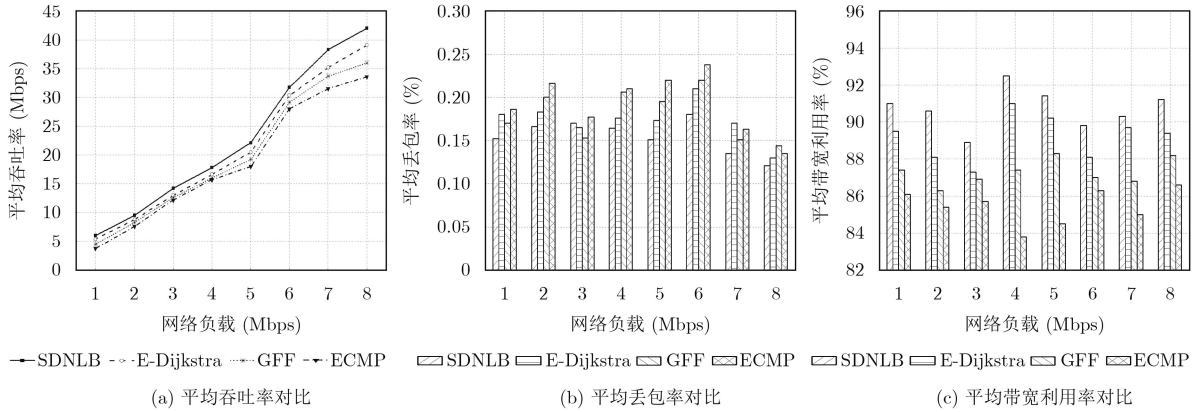
服务器编号	2	4	6	8	10	12	14	16
SDNLB	22.65	22.53	22.43	22.31	22.20	22.58	22.48	22.11
E-Dijkstra	23.39	23.34	23.32	23.29	23.26	23.50	23.36	23.29
GFF	23.40	23.46	23.45	23.48	23.46	23.37	23.34	23.54
ECMP	23.79	23.70	23.66	23.60	23.59	23.75	23.68	23.56

表 4　服务器平均负载

服务器编号	2	4	6	8	10	12	14	16
SDNLB	0.71	0.71	0.70	0.71	0.68	0.69	0.73	0.72
E-Dijkstra	0.75	0.74	0.75	0.76	0.74	0.74	0.76	0.77
GFF	0.74	0.75	0.78	0.78	0.77	0.74	0.77	0.78
ECMP	0.86	0.86	0.87	0.89	0.87	0.87	0.85	0.87

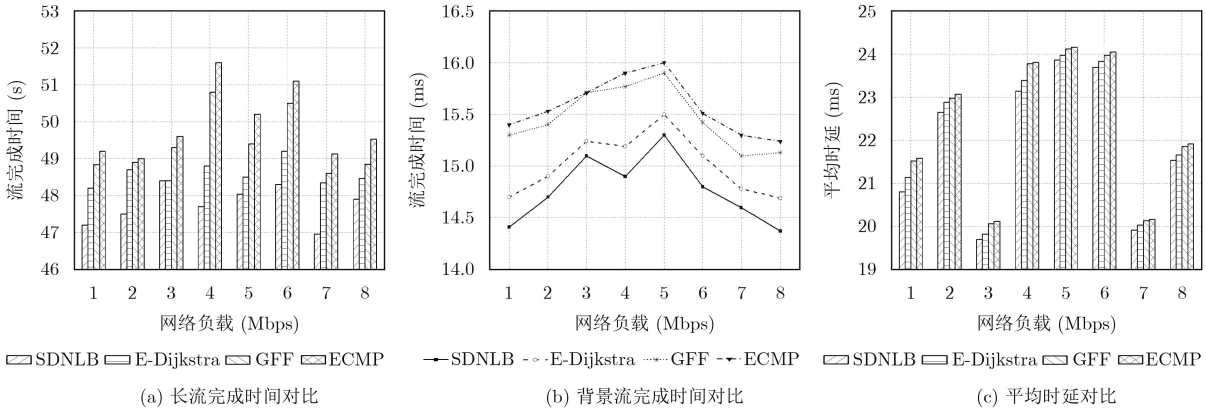
都达到了峰值，表明此阶段网络中出现了较严重的拥塞现象导致交换机端口缓存队列溢出造成大量数据包被丢弃。这也说明了流量具有局部突发的特性，与文献[17]的研究结论一致。图2(c)给出了每种算法的平均带宽利用率，由图中数据可知，SDNLB能够根据服务器的运行状态和网络链路的负载情况选择最优转发路径，有效地改善了部分链路过载或空闲的问题，大幅提高了网络带宽利用率。而E-Dijkstra算法仅以链路和交换机中传输的字节数作为权值计算的依据，忽略了流速对于链路带宽和交换机缓存队列的影响，因此当网络中存在混合流时，算法性能出现了下降。GFF算法虽然能够对长流进行重路由传输，但是无法保证所选路径为负载最轻的路径。ECMP算法采用了哈希运算来分配转发路径，当哈希值重叠时，则将多个流分配到一个输出端口，这样会导致路由分布不均匀，因此带宽利用率不高。

图3(a)和图3(b)分别展示了长流和背景流的完成时间。在图3(a)中，当网络负载处于4~6 Mbps范围内时，ECMP算法和GFF算法的流完成时间变化幅度较大，表明网络中出现了拥塞链路，网络性能下降。由于SDNLB采用最优转发路径策略进行流转发，因此在不同的负载条件下，流完成时间均保持最少。而E-Dijkstra算法在计算路径时对路径中的某条链路是否处于拥塞状态并不敏感，在多流调度时，完成时间会相应增大。在图3(b)中，当网络负载处于5 Mbps时，每种算法的流完成时间均达到峰值，这验证了网络中不同类型的数据分组队列是相互影响的。图3(c)为背景流的平均时延。同样可从图中看出，在网络负载由3 Mbps增加到4 Mbps时，每种算法的平均时延快速上升；在网络负载由6 Mbps增加到7 Mbps时，每种算法的平均时延又出现大幅回落，说明平均时延大小与网络的拥塞程度是密切相关的。



(a) 平均吞吐率对比 (b) 平均丢包率对比 (c) 平均带宽利用率对比

图 2 不同算法的网络性能指标对比



(a) 长流完成时间对比 (b) 背景流完成时间对比 (c) 平均时延对比

图 3 不同算法的流完成时间、平均时延对比

5 结束语

针对当前服务器集群负载均衡技术存在的不足之处，本文提出了SDNLB机制。该机制借助SDN具有的全局网络拓扑感知和灵活的数据转发能力来获取网络状态的变化，同时利用SNMP协议实时监测各服务器的负载状态，通过权值计算的方式选出权重最高的目标服务器，在此基础上，采用最优转发路径算法进行流量调度，从而保证服务器集群能

够具有更高的利用率和处理性能。文中第4节的实验结果表明，相比于E-Dijkstra算法和GFF算法，SDNLB将服务器集群整体的平均负载分别降低了6%和8%，将网络平均吞吐率分别提高了7%和16%，同时提升了平均带宽利用率并缩短了流的完成时间和平均时延。在下一步工作中，将继续完善SDNLB对网络流量的调度机制，并在其他类型的大规模网络拓扑环境中进行性能验证和分析。

参 考 文 献

[1] GHOMI E, RAHMANI A, and QADER N. Load-balancing algorithms in cloud computing: A survey[J]. *Journal of Network and Computer Applications*, 2017, 88(12): 50–71. doi: 10.1016/j.jnca.2017.04.007.

[2] SHARMA G and BUSCH C. A load balanced directory for distributed shared memory objects[J]. *Journal of Parallel and Distributed Computing*, 2015, 78(4): 6–24. doi: 10.1016/j.jpdc.2015.02.002.

[3] ILCHOL P, QIAO Baiyou, SHEN Muchuan, *et al*. An efficient load balancing approach for N-hierarchical web server cluster[J]. *Wuhan University Journal of Natural Sciences*, 2015, 20(6): 537–542. doi: 10.1007/s11859-015-1130-9.

[4] YANG Juipin. Elastic load balancing using self-adaptive replication management[J]. *IEEE Access*, 2017, 5(99): 7495–7504. doi: 10.1109/ACCESS.2016.2631490.

[5] SHEIKHI S and BABAMIR S. A predictive framework for load balancing clustered web servers[J]. *The Journal of Supercomputing*, 2016, 72(2): 588–611. doi: 10.1007/s11227-015-1584-8.

[6] MAO Qilin and SHEN Weikang. A load balancing method based on SDN[C]. The 7th International Conference on Measuring Technology and Mechatronics Automation, Nanchang, China, 2015: 18–21. doi: 10.1109/ICMTMA.2015.13.

[7] TRESTIAN R, KATRINIS K, and MUNTEAN G. OFLoad: An OpenFlow-based dynamic load balancing strategy for datacenter networks[J]. *IEEE Transactions on Network and Service Management*, 2017, 14(4): 792–803. doi: 10.1109/TNSM.2017.2758402.

[8] 李龙, 付斌章, 陈明宇, 等. Nimble: 一种适用于OpenFlow网络的快速流调度策略[J]. 计算机学报, 2015, 38(5): 1056–1068. doi: 10.3724/SP.J.1016.2015.01056.

LI Long, FU Binzhang, CHEN Mingyu, *et al*. Nimble: A fast flow scheduling strategy for OpenFlow networks[J]. *Chinese Journal of Computers*, 2015, 38(5): 1056–1068. doi: 10.3724/SP.J.1016.2015.01056.

[9] AL-FARES M, RADHAKRISHNAN S, RAGHAVAN B, *et al*. Hedera: Dynamic flow scheduling for data center networks[C]. NSDI'10 Proceedings of the 7th USENIX conference on networked systems design and implementation, San Jose, USA, 2010: 281–296.

[10] 蔡岳平, 王昌平. 软件定义数据中心网络混合路由机制[J]. 通信学报, 2016, 37(4): 44–52. doi: 10.11959/j.issn.1000-436x.2016071.

CAI Yueping and WANG Changping. Software defined data center network with hybrid routing[J]. *Journal on Communications*, 2016, 37(4): 44–52. doi: 10.11959/j.issn.1000-436x.2016071.

[11] 覃匡宇, 黄传河, 刘柯威, 等. 基于多路广播树的SDN多路径路由算法[J]. 计算机科学, 2018, 45(1): 211–215. doi: 10.11896/j.issn.1002-137X.2018.01.037.

TAN Kuangyu, HUANG Chuanhe, LIU Kewei, *et al*. Multipath routing algorithm in software defined networking based on multipath broadcast tree[J]. *Computer Science*, 2018, 45(1): 211–215. doi: 10.11896/j.issn.1002-137X.2018.01.037.

[12] LIAO Lingxia and LEUNG VC. LLDP based link latency monitoring in software defined networks[C]. The 12th International Conference on Network and Service Management, Montreal, Canada, 2016: 330–335. doi: 10.1109/CNSM.2016.7818442.

[13] RUBIN I and ZHANG Runhe. Max-min utility fair flow management for networks with route diversity[J]. *International Journal of Network Management*, 2010, 20(6): 361–381. doi: 10.1002/nem.740.

[14] YEN J Y. Finding the K shortest loopless paths in a network[J]. *Management Science*, 1971, 17(11): 712–716. doi: 10.1287/mnsc.17.11.712.

[15] JIANG J R, HUANG H W, LIAO J H, *et al*. Extending Dijkstra's shortest path algorithm for software defined networking[C]. The 16th Asia-Pacific Network Operations and Management Symposium, Hsinchu, China, 2014: 1–4. doi: 10.1109/APNOMS.2014.6996609.

[16] ZHANG Zhe, BOCKELMAN B, CARDER D, *et al*. Lark: An effective approach for software-defined networking in high throughput computing clusters[J]. *Future Generation Computer Systems*, 2017, 72(7): 105–117. doi: 10.1016/j.future.2016.03.010.

[17] CHEN Yingying, JAIN S, ADHIKARI V, *et al*. A first look at inter-data center traffic characteristics via Yahoo! datasets[C]. IEEE INFOCOM, Shanghai, China, 2011: 1620–1628. doi: 10.1109/INFCOM.2011.5934955.

于天放：男，1980年生，博士生，研究方向为软件定义网络.

芮兰兰：女，1979年生，博士，副教授，研究方向为网络和业务质量管理、泛在网络、大数据等.

邱雪松：男，1973年生，博士，教授，研究方向为网络管理与通信软件.