计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200400131

带宽和时延受限的流媒体服务器集群负载均衡机制

郑增乾[1]　王　锟[1]　赵　涛[2]　蒋　维[3]　孟利民[1]
1 浙江工业大学信息工程学院　杭州 310000
2 浙江省通信产业服务有限公司　杭州 310000
3 浙江树人大学信息科技学院　杭州 310000
(2111803022@zjut.edu.cn)

摘　要　流媒体服务器集群的整体负载能力很大程度上受其服务时延和带宽负载均衡程度的影响。因此如何提高服务实时性和均衡带宽负载是提升流媒体服务器集群服务能力的关键。为此,提出了一种带宽和时延受限的流媒体服务器集群负载均衡机制。该机制通过将服务器带宽和任务带宽的离散化、区间化,构建服务器与任务状态集,再利用遗传算法离线计算并存储各个状态下的最佳分配方案,使得在有效地将不同带宽需求的任务分配到各个服务器上优化集群负载的同时,加快在线任务分配方案的计算速度,提高时效性。仿真结果显示,该机制能够在拥有与轮询算法、最小连接数算法相似的计算时延的基础上,有效均衡带宽负载,降低失败任务数,从而提升整体服务质量和能力。

关键词:流媒体服务;服务器集群;负载均衡;服务质量;离线计算;遗传算法
中图法分类号　TP301

Load Balancing Mechanism for Bandwidth and Time-delay Constrained Streaming Media Server Cluster

ZHENG Zeng-qian[1],WANG Kun[1],ZHAO Tao[2],JIANG Wei[3] and MENG Li-min[1]
1 College of Information Engineering,Zhejiang University of Technology,Hangzhou 310000,China
2 Zhejiang Communication Industry Service,Co.,Ltd.,Hangzhou 310000,China
3 College of Information Science and Technology,Zhejiang Shuren University,Hangzhou 310000,China

Abstract　Overall load capacity of streaming media server cluster is largely affected by its service delay and bandwidth load balancing. Therefore,how to improve the real-time capability of service and balance the bandwidth load are the keys to improve the streaming media server cluster service capabilities. This paper proposes a load balancing mechanism for bandwidth and time-delay constrained streaming media server cluster. Through discretizing bandwidth of server and task,the mechanism builds the server and task state sets. And it uses genetic algorithms to calculate and store the optimal allocation scheme in each state offline to speed up the online task assignment scheme calculation while effectively allocating tasks with different bandwidth requirements to each server to optimize the cluster load. Results of simulation show that the mechanism can effectively balance the bandwidth load and reduce the number of failed tasks on the basis of having a calculation delay similar to the round-robin algorithm and least connections algorithm,thereby improving the overall service quality and ability.

Keywords　Streaming service,Server cluster,Load balancing,Quality of service,Calculate off-line,Genetic algorithm

1　引言

随着网络技术的迅速发展与逐渐成熟,互联网在人们的生活中已不可或缺,加之移动网络和移动设备的广泛普及,互联网用户数量呈现不断增长的状态。在互联网用户数量激增的当下,人们对网络信息的需求也逐渐变得多元化。除了传统的文本信息和图像信息之外,人们同样希望以流媒体的形式获得音视频等多媒体信息,并且音视频流量在总移动数据流量中的占比超过50%[1-2]。同时,用户的服务请求往往不可预测。因此,常常会出现服务器在短时间内收到大量并发流媒体服务请求的情况。此时,在传统的单台服务器的工作模式下,大量请求在任务池里排队等待处理,服务器会被动降低其服务能力,往往无法及时响应所有用户的服务请求,最终导致无法为用户提供良好的用户体验[3]。

集群技术是解决上述问题的常用手段[4]。通过将多台独立服务节点连接在一起,使不同服务节点同时协同工作,组成

到稿日期:2020-04-28　返修日期:2020-06-23　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金(61871349);浙江省自然科学基金(LQ19F010013,LY18F010024);2019年金华市科技计划项目(公益类)(2019-4-176)
This work was supported by the National Natural Science Foundation of China(61871349),Natural Science Foundation of Zhejiang Province,China(LQ19F010013,LY18F010024) and Science and Technology Program of Jinhua in 2019(2019-4-176).
通信作者:孟利民(mlm@zjut.edu.cn)

多节点松耦合的集群系统结构，从而高效完成任一单一节点无法完成的任务。如何保证被分配到具体服务节点的请求被及时有效地处理是集群技术的核心问题之一。负载均衡技术作为解决该问题的关键技术之一，其目的是平衡集群中各服务器的负载，使各服务器能够在需要的时候发挥出其最大的带负载性能，从而整体提升集群服务的质量与能力。

在同一集群中，一方面服务器的计算能力、内存容量、带宽能力等性能不一定全部相同，另一方面服务请求的带宽需求、实时性需求等也存在差异。负载均衡策略需要综合考虑集群中各服务器的能力以及服务请求对各方面性能的需求，才能充分发挥服务器的作用，规避负载倾斜。

高并发的流媒体服务具有高实时性、高带宽需求且高实时性很大程度上依赖于高带宽需求的特征[5]，因此负载均衡机制如何满足其高实时性、高带宽需求以降低请求失败率、提高服务质量和能力是关键。因此，本文提出一种带宽和时延受限的流媒体服务器集群负载均衡机制，以满足高并发流媒体服务请求对高实时性、高带宽的需求，从而提升带宽和时延受限的流媒体集群的服务质量和能力。

2 相关工作

2.1 静态、动态负载均衡机制

负载均衡技术的关键在于均衡的任务调度。目前，国内外的学者们已提出各种任务调度算法及其改进算法，根据其是否能以服务器的实时状态做出相应的动作大致上可以分为两大类：静态负载均衡和动态负载均衡[6]。常见的静态负载均衡算法如轮询算法、加权轮询算法、散列算法等。此类算法的优势在于实现难度低，但其以某一事先设定的固定规则进行任务调度分配，未能考虑到服务器的实际负载状况，容易引起负载倾斜。常见的动态负载均衡算法如最小连接数算法、一致性哈希算法[7]、动态反馈算法[8]等。此类算法没有考虑到不同服务器之间的性能差异和请求任务的负载需求，不能准确获得真实的负载情况，且其对负载和分配策略的计算需要额外的开销。

2.2 负载指标及其反馈周期的选择

负载指标选择的好坏是直接影响负载均衡效果的重要因素。在负载指标选择上，有选择响应时间[9]、CPU 利用率和内存利用率[10]、连接任务数[11]、综合考虑各指标[12]等均衡方法。在指标的选择上没有统一的最佳指标，应当根据集群提供的服务类型做相应变化。

Wang 等[13]提出了一种通过构建服务器 M/M/1 排队模型，以任务的平均排队时长和平均等待时长作为系统指标计算负载，但其计算较繁琐，实时性不足。Wang 等[14]提出的流媒体集群负载均衡调度算法，根据节点任务数变化对负载反馈周期进行动态修改，并对节点负载水平进行分类，但当反馈周期过小时，服务器负载的计算、分类过于频繁，导致服务器压力增加；当反馈周期过大时服务器无法处理大量的并发任务。Li 等[15]同样在负载均衡的反馈周期上进行研究并提出了自适应的方法，但其反馈周期的调整同样需要计算，具有滞后性，且其计算以请求数量为指标，未考虑到不同请求带来的负载差异，容易引起负载倾斜。

2.3 与智能计算结合的负载均衡机制

学者们还提出了许多注重任务调度的智能算法[16-18]，其实现较为复杂，时间复杂度高，且有陷入局部最优解的风险。

Li 等[19]提出了一种基于改进粒子群算法的云计算负载均衡算法，通过计算资源的利用率来定义负载因子并将其作为粒子群算法的适应度函数，但其计算效率一般，每次任务到来时都需要花费较多时间计算任务分配方案，对服务器的响应时间有一定影响，无法满足流媒体服务对高实时性的需求。

综上，静态负载均衡机制因缺少对服务器负载及任务实际状况的考虑易引起负载倾斜；而动态负载均衡机制没有考虑到服务器的性能差异和请求任务的负载需求，且其动态计算往往需要较大的资源开销，尤其是在结合智能计算后，动态负载均衡算法会影响服务的实时性；负载指标的选择没有最佳标准，需要根据实际情形选定；反馈周期往往需要对负载指标的时效性和计算资源的占用进行权衡。基于此，本文提出了一种负载均衡机制，在动态负载均衡机制的基础上，通过离线计算和在线分配相结合的方式，在离线计算时引入智能计算，从而降低计算过程对在线分配实时性的影响，减少在线分配的计算开销，并将服务器的性能差异、任务负载的需求纳入离线计算考虑范围内；在负载指标选择上，因流媒体高带宽需求的特点，将带宽作为主要负载指标；在反馈周期上，通过引入任务管理器，当服务请求下发和服务结束时，在任务管理器上维护服务器状态信息，以达到在保证负载指标高实时性的同时，降低对计算资源占用的目的。

3 流媒体集群带宽负载均衡机制

3.1 流媒体集群带宽负载均衡机制

根据流媒体服务具有的高实时性、高带宽需求的特点，本文提出了一种带宽和时延受限的流媒体服务器集群负载均衡机制。该机制通过对任务分配优先度的方式优先应答更为紧急的服务请求，通过离线计算的方式减少在线计算分配方式所需的时间，通过对服务请求的带宽需求和集群带宽能力进行分析并结合遗传算法，计算当前服务器带宽能力下大量并发服务请求到来时的任务分配策略。

该机制结合了贪婪算法的基本思想：在任务调度时，从任务池队列中选取一定数量（N）最先到达的任务，根据任务状态（m_j）、服务器状态（$Server_k$）计算当前最优任务分配顺序及分配执行的服务器，即获得当前局部最优解（c_j）；再从当前局部状态出发，求解下一状态局部最优解，最终获得全局最优解。该机制可以分为两个阶段：1）离线构建状态表阶段，系统预先计算各个系统状态（S_i）下的最优解，以键（局部状态）值（调度方法）对的形式构建状态表；2）在线调度阶段，系统监测服务器集群状态以及任务状态，构建出当前局部状态，再根据当前状态选择预先计算好的局部最优解进行任务分配。

3.2 离线构建状态表阶段

该阶段的目标是构建任务分配状态表。如图 1 所示，该状态表由任务生成模块、任务评估模块、虚拟服务器集群状态生成模块、系统状态组合模块、分配方案计算模块和任务分配状态表组成。任务生成模块负责批量生成各种带宽需求的任务。任务评估模块负责拉取任务并评估其带宽需求进而生成任务状态。虚拟服务器集群状态生成模块负责生成各个虚拟服务器状态。系统状态组合模块负责拉取服务器集群状态和任务状态组合形成系统状态。分配方案计算模块拉取系统状态，计算该状态下的最佳分配方案（c_j），并记录更新于任务分

配状态表中。具体执行步骤如下：

(1)任务生成模块不断批量生成各种带宽需求的任务，供下一模块拉取。

(2)假设集群中服务器数量为 K，则任务评估模块从任务生成模块拉取 K 个任务并评估该任务的带宽需求，构建出待分配任务状态 m_{j2}。

(3)虚拟服务器集群状态生成器生成服务器带宽使用情况，构建服务器状态 $Server_k$。

(4)系统状态组合模块从任务评估模块拉取任务状态 m_{j2}，从虚拟服务器集群状态生成器拉取服务器状态 $Server_k$，并组合形成系统状态 S_i，供下一模块使用。

(5)分配方案计算模块从上一模块拉取系统状态，并利用遗传算法计算该状态下的最优分配方案 c_j。

(6)分配方案计算模块根据计算结果，建立 S_i 到 c_j 的映射关系。

(7)任务分配状态表拉取 S_i 到 c_j 的映射关系并进行记录更新。

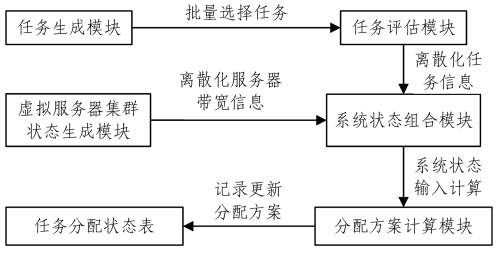


图 1 构建任务分配状态表的架构示意图

Fig. 1 Architecture diagram of generate mission distribute table

3.3 在线调度阶段

图 2 为在线分配任务架构示意图。

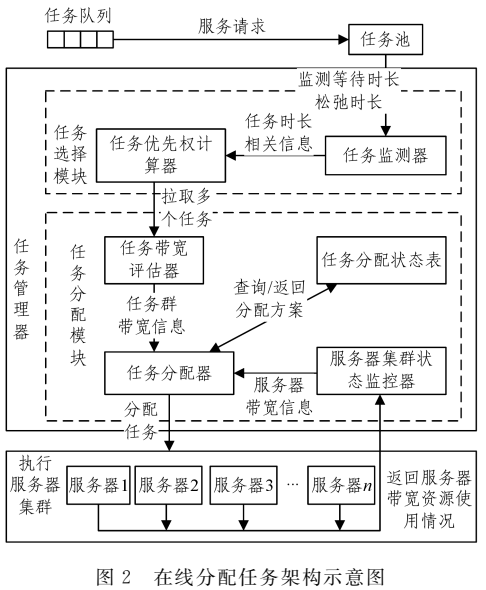


图 2 在线分配任务架构示意图

Fig. 2 Architecture diagram of distribute missions on-line

该调度阶段通过任务管理器管理任务选择顺序和分配方式来达到减少失败数、降低任务等待时长与任务松弛时长比、均衡服务器集群负载的目的。其步骤具体如下：

(1)任务监测器从任务池中拉取一定数量的任务，监测这些任务的等待时长和松弛时长，从而构建出(m_{j1})作为任务选择器的输入。

(2)任务优先权计算器根据输入的待选择任务状态(m_{j1})，计算各任务执行的优先度，并从中挑选 K 个优先度最高的任务输出到下一模块。

(3)任务带宽评估器接收来自上一模块的任务，并评估该任务的带宽需求，构建出待分配任务状态(m_{j2})。

(4)服务器集群状态监控器能监控集群中服务器带宽的使用情况，在任务到达服务器和任务完成时更改其状态以维护集群状态信息 $Server_k$。

(5)任务分配器从任务带宽评估器获取 m_{j2}，从服务器集群状态监控器获取 $Server_k$，结合两者构建出任务分配阶段系统状态 S_i，并根据此状态查询离线生成的任务分配状态表获得最佳分配方案，再以此方案分配任务到具体服务器执行。

综上，该阶段以任务池任务和服务器状态为输入，以此时的最佳分配方案为输出，动态地分配任务到各服务器执行。该分配算法的流程图如图 3 所示。

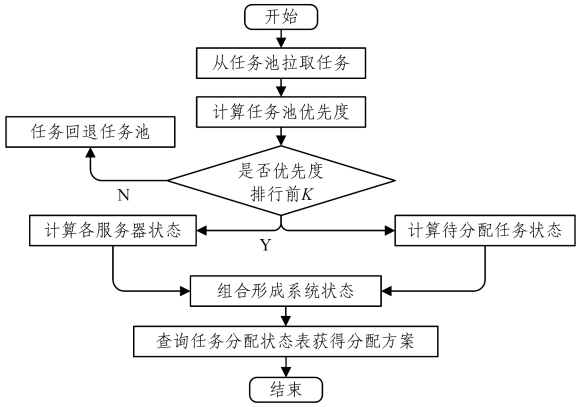


图 3 在线任务分配方案的计算流程

Fig. 3 Process of distribute missions solution on-line

3.4 任务优先度计算

一个任务的优先度(P)应当由该任务的松弛时长、已等待时长决定，且应满足以下 3 个特性：

(1)优先度应与松弛时长呈负相关；

(2)优先度应与等待时长呈正相关；

(3)优先度应与松弛时长和等待时长的差值呈负相关，且差值越小则优先度增长越快。

由此，任务的优先度可由式(1)计算：

$$P=\frac{a}{t_{except}-t_{wait}}+b*t_{wait}-c*t_{except} \tag{1}$$

其中，t_{except} 表示任务的松弛时长，即任务最大可等待时长；t_{wait} 表示任务在任务池中已等待时长；任务剩余可等待时长则由 $t_{except}-t_{wait}$ 表示；a，b，c 为计算系数，分别表示任务优先度 P 对任务剩余可等待时长、任务已等待时长和任务松弛时长的看重程度，值越大表示对相应指标越重视。

4 离线构建状态表

本文提出的针对带宽和时延受限情景的流媒体服务器集群负载均衡机制的核心在于离线构建任务分配状态表，状态表的构建方式将直接影响该机制的性能。通过对服务请求的带宽需求和集群带宽能力进行监测并结合遗传算法，计算当前任务集群状态下的最佳任务分配方式。

4.1 服务质量与能力

当一批流媒体请求陆续到达后，可以从以下几个方面对

服务器集群的整体服务质量与能力进行评价：

(1)带宽利用率方差在时间上的均值为$\bar{\sigma^2}$，即各个服务器带宽利用率的均衡情况：

$$\sigma^2=\frac{1}{k}\sum_{k=1}^{K}(r_k-\bar{r})^2 \tag{2}$$

其中，K 表示集群中的服务器数量；r_k 表示第 k 台服务器的带宽利用率；$\bar{r}$ 表示平均带宽利用率，可由式(3)计算：

$$\bar{r}=\frac{1}{K}\sum_{k=1}^{K}r_k \tag{3}$$

其中，$\bar{\sigma^2}$ 为 σ^2 在不同时刻上采样的均值。

(2)任务等待时长与松弛时间之比的均值$\overline{r_wait}$：

$$\overline{r\_wait}=\frac{1}{L}\sum_{l=1}^{L}r\_wait_l \tag{4}$$

$$r\_wait_l=\frac{t_{wait}^l}{t_{except}^l} \tag{5}$$

其中，t_{wait}^l 表示任务请求 l 已等待被分配的时长；t_{except}^l 表示任务请求 l 可等待的松弛时长。

(3)请求失败数Num_{fail}在总请求数Num_{all}中的占比R_{fail}：

$$Num_{fail}=Num_{fail1}+Num_{fail2} \tag{6}$$

$$Num_{fail1}=\sum_{l=1}^{L}flag_{l1} \tag{7}$$

$$Num_{fail2}=\sum_{l=1}^{L}flag_{l2} \tag{8}$$

$$R_{fail}=\frac{Num_{fail}}{Num_{all}} \tag{9}$$

请求失败分为两种情况：第一种情况为，任务请求到达时间到任务分配到具体服务器的时间间隔大于请求可等待时间；第二种情况为任务请求所需带宽大于服务器所能提供带宽，即服务器无法完成任务。当出现以上任意一种情况时，认定为任务请求失败。$flag_{l1}$值为 1 时表示任务请求 l 发生第一种情况，$flag_{l2}$值为 1 时表示任务请求 l 发生第二种情况，其公式如下：

$$flag_{l1}=\begin{cases}0, & t_{wait}^l<t_{except}^l\\ 1, & t_{wait}^l\geqslant t_{except}^l\end{cases} \tag{10}$$

$$flag_{l2}=\begin{cases}0, & band_{need}^l<band_{server}^l\\ 1, & band_{need}^l\geqslant band_{server}^l\end{cases} \tag{11}$$

因此，当某一批多媒体任务请求到达时，服务器集群的 $QCoS$ 可以表示为带宽利用率方差在时间上的均值$\bar{\sigma^2}$、任务等待时长与松弛时间之比的均值$\overline{r_wait}$、请求失败数Num_{fail}在总请求数Num_{all}中的占比R_{fail}的函数，其公式如下：

$$QCoS(C_i)=F(\bar{\sigma^2},\overline{r\_wait},R_{fail})=\frac{\alpha}{e^{\bar{\sigma^2}}}+\frac{\beta}{e^{\overline{r\_wait}}}-\gamma*R_{fail} \tag{12}$$

其中，α,β,γ 分别表示集群服务能力与质量对带宽利用率方差、任务等待时长与松弛时间之比的均值以及失败任务数占比的看重程度，值越大表示对相应指标越重视。因此，负载均衡策略C_i的目标可以表示为：

$$\max\ QCoS(C_i) \tag{13}$$

4.2 任务分配阶段系统状态S_i和回报函数R

任务分配阶段系统状态(S_i)主要由两方面因素决定：服务器集群状态($Server_k$)、正在分配的批量任务状态(m_{j2})，即：

$$S_i=(Server_k,m_{j2}) \tag{14}$$

服务器集群状态$Server_k$可以由一个多维二元组$\langle b_k,r_k\rangle$表示。其中，k 代表维数，即第 k 台服务器；对于具体某一台服务器，其总带宽由b_k表示，为一固定值；其带宽使用率用r_k表示。

待分配任务状态m_{j2}可以由一个多维一元组$\langle mb_j\rangle$表示。mb_j表示某一具体任务所需带宽。

任务分配阶段回报函数。服务器能提供的 $QCoS$ 与此阶段的请求失败数占比、分配后带宽利用率方差呈负相关。故该阶段的回报函数(R)可用式(15)表示：

$$R=\tau R_\sigma-\varphi R_{f2} \tag{15}$$

其中，R_σ为带宽利用率方差的回报函数，R_{f2}为请求失败数占比的回报函数。其计算方式分别如式(16)、式(17)所示：

$$R_\sigma=\frac{1}{e^{\bar{\sigma^2}}} \tag{16}$$

$$R_{f2}=\frac{Num_{fail2}}{Num_{all}} \tag{17}$$

4.3 构建状态表

每台服务器的带宽以及任务带宽需求都是连续值，所以其组成的状态在数量上都是无限的，导致状态表的大小也为无限大。因此，可对带宽和时长做离散化、区间化处理以达到减少状态数量、减小状态表大小的目的。具体离散化、区间化的方法如下：设需作离散化、区间化处理的量为 ω，区间大小为 d，处理后的输出量为 μ，则 $\mu=(\omega/d)+(d/2)$。

在各个连续量完成离散化、区间化后，通过遗传算法计算可得当前系统状态下的最佳任务分配方案。最佳任务分配方案序列可表示为($c_{m1},c_{m2},\cdots,c_{mn}$)，$c_{mj}\in\{Server_1,\cdots,Server_k\}$。其中，$c_{mj}$表示任务$m_j$被分配到的服务器。本文以分配方案为表现型进行染色体二进制编码。一条染色体包含 n 个基因位，每个基因位由一个 p 比特二进制数表示，p 满足 $2^p>k$，k 为服务器台数。将$2^p/k$取整记为 x，则 $x*c_{mj}$ 的 p 比特二进制表示为任务m_j被分配到的服务器的编号的二进制编码结果，便可将其填入相应基因位。故编码时，将分配方案中的服务器编号转化为相应二进制数并填入相应基因位即可完成编码。解码为编码的逆过程，读取基因位上的二进制数转化为十进制，再除以 x 后取整即为对应任务分配到的对应服务器编号。当$2^p/k$不是整数时，解码出的服务器编号可能会出现大于最大编号的情况，此时将其解码为最大服务器编号即可。同时，选用的遗传算法以任务状态和服务器状态组成的系统状态S_i作为进化环境，以回报函数 R 的值作为个体对环境的适应性，以轮盘赌的方式进行个体选择，以单点交叉作为个体间交叉方式，以单点基因变异作为个体变异方式，进行任务分配策略的计算。算法 1 描述了生成任务分配阶段状态表的流程。

算法 1 任务分配阶段状态表的生成

输入：任务带宽需求集合MB_J，服务器集群状态集合$SERVER_K$，权重系数 τ 和 φ，进化代数 δ，交叉概率 pCross，变异概率 pMutation

输出：任务分配状态表 SolutionTable

```
1. for mb_j in MB_J do
2.     for Server_k in SERVER_K do
3. 组合(Server_k, mb_j)形成S_i2
4.         初始化 population
5.         while δ>0 do
```

6. 解码 individual 获得 solution
7. $R_\sigma \leftarrow 1/e^{\bar{\sigma^2}}$，$R_{f2} = Num_{fail2}/Num_{all}$
8. $R \leftarrow \tau R_\sigma - \varphi R_{f2}$ //计算回报函数 R
9. 以 R 为适应度保存各 individual 适应度
10. for individual in population do
11. 种群选择操作
12. 以 pCross 概率进行单点交叉操作
13. 以 pMutation 概率进行单点变异操作
14. 保存适应度最大的 individual
15. end for
16. 添加新 individual 到 population
17. $\delta \leftarrow \delta - 1$
18. end while
19. 解码最佳 individual 获得 solution
20. 保存 S_{i2} → solution 映射到 SolutionTable
21. end for
22. end for
23. return SolutionTable

5 实验结果与性能分析

本文采用 JavaSE 平台编写管理服务器和服务器集群的仿真程序，系统采用一个虚拟管理服务器和 3 个虚拟服务器组成的虚拟服务器集群，将任务信息、服务器信息存储于 Redis 内存中来维护任务池和仿真任务的运行。仿真程序运行的计算机配置如下：双路 Core(TM) i5-4258U @2.4 GHz，内存 8 GB。3 个虚拟服务器的带宽能力分别配置为 400 M，600 M，800 M。

实验 1　不同任务优先度参数下的失败率对比

为比较不同任务优先度参数下因等待超时引起的任务失败率，通过每 300 ms 向集群发起 3 个任务请求，每 100 ms 从等待任务中选取优先度最高的任务运行，进而计算任务失败率。其中，每组参数下重复向集群发送3000个任务请求 3 次，且任务的松弛时长的十分之一服从均值为 60 的泊松分布。实验结果如表 1 所列，可以看出，$a=15, b=8, c=5$ 和 $a=5, b=8, c=5$ 时任务的平均失败率相近且最低，但当 $a=5, b=8, c=5$ 时 3 次重复实验的失败率相差较大，较为不稳定。故在后续实验中，任务优先度的参数选为 $a=15, b=8, c=5$。

表 1　不同任务优先度参数下的失败率

Table 1　Failure rate under different priority parameters

a	b	c	Fail num_1	Fail num_2	Fail num_3	Average Failrate/%
5	5	8	78	124	131	3.7
5	8	5	47	98	101	2.7
10	5	8	68	88	137	3.3
10	8	5	139	133	114	4.3
15	5	8	104	113	105	3.6
15	8	5	87	88	58	2.6
20	5	8	78	45	148	3.0
20	8	5	119	67	103	3.2

实验 2　不同 *QCoS* 参数下任务运行情况的对比

首先，并非所有的 *QCoS* 参数都会影响离线阶段分配策略的计算。离线计算过程中回报函数 R 仅与带宽利用率方差和请求失败数占比相关，因此其仅与计算 *QCoS* 的参数中的 α, γ 相关。β 的选择仅影响 *QCoS* 值，不会对离线计算分配方案产生影响。本实验以 10 missions/s 速率向集群发送 3000 个服务请求，对比在不同的 α, γ 下的带宽利用率方差在时间上的均值和任务失败率。实验结果如表 2 所列，当 $\alpha=10, \gamma=50$ 时，任务失败率略高；当 $\alpha=50, \gamma=100$ 时，带宽利用率方差在时间上的均值略高，其他参数下的负载均衡结果相近；当 $\alpha=10, \gamma=100$ 时，带宽利用率方差和任务失败率都处于平均水平，故在后续实验中将 *QCoS* 的计算参数 α 和 γ 分别设置为 10 和 100。关于参数 β 的选取，实质上 β 仅影响服务器负载质量和能力的判定而不影响负载均衡策略的执行。为避免选取的 β 值过大而削弱了 $\bar{\sigma^2}$ 和 R_{fail} 的作用或 β 值过小而忽略了 $\overline{r\_wait}$，故在后续实验中设置 β 值为 5。

表 2　不同 *QCoS* 参数下的负载均衡结果

Table 2　Results of load balance under different *QCoS* parameters

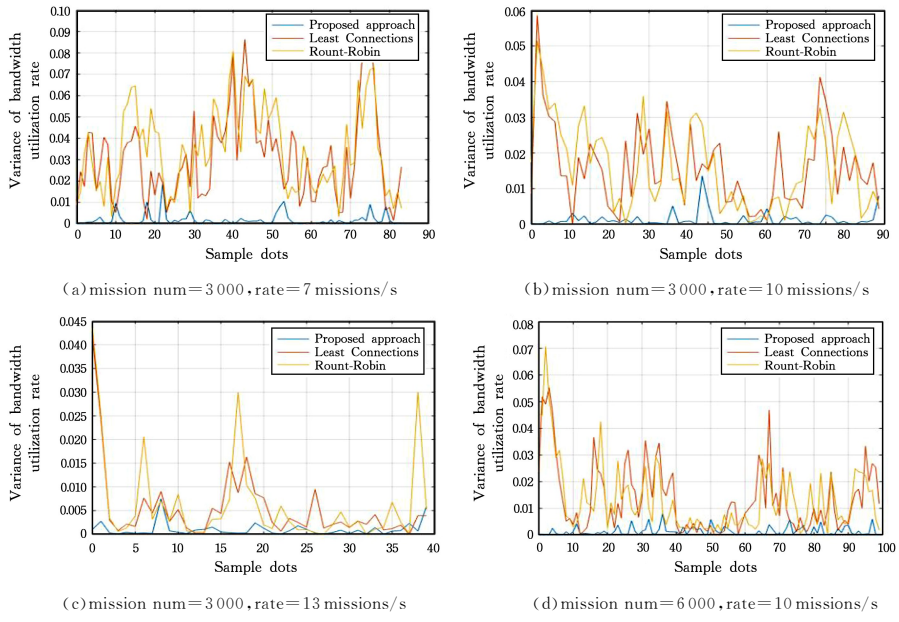
α	γ	$\bar{\sigma^2}$	Fail num
10	5	0.001 353	97
10	50	0.001 396	115
10	100	0.001 226	100
50	5	0.001 327	101
50	50	0.001 102	98
50	100	0.001 711	103
100	5	0.001 170	100

实验 3　基于本文算法的负载均衡机制和基于轮询算法、最小连接数算法的负载均衡机制的效果比较

通过采用本文提出的算法，以不同的任务到达速率向管理服务器发送不同数量的任务。其中 6 M 到 14 M 的较低带宽需求任务占总任务请求的 90%，40 M 到 60 M 的较高带宽需求任务占总任务请求的 10%。对带宽进行量化时，任务带宽量化区间大小取 4 M，服务器带宽量化区间大小取 10 M。在计算任务优先度时，计算参数配置如下：$a=15, b=8, c=5$。在计算 *QCoS* 时，计算参数配置如下：$\alpha=10, \beta=5, \gamma=100$。

本实验以 10 missions/s 速率向集群发送 3000 个服务请求，对基于轮询算法[20]、最小连接数算法[21]和本文算法的负载均衡机制进行仿真并对结果进行分析。其中，基于轮询算法的负载均衡机制不关心服务器的当前负载和请求的资源需求差异，直接将服务请求按顺序轮流地分配到服务器上，通过向所有服务器平均分配服务请求，以期达到均衡各服务器负载、提升整体负载能力的目的；基于最小连接数算法的负载均衡机制则动态选取当前连接服务最少的一台服务器，将服务请求分配到该服务器上执行，以尽可能地均衡各服务器的资源利用率，达到提升整体负载能力的目的。仿真开始后，每 5 s 记录一次服务器的负载情况即带宽占用情况。

带宽使用率在不同时刻的方差可以反映集群中服务器的带宽负载均衡状况。仿真结果的带宽负载均衡状况如图 4(b)所示。图中横坐标为采样点，纵坐标为带宽使用率的方差。观察该图可知，当以 10 missions/s 速率向集群发送 3000 个服务请求时，基于本文算法的集群的带宽负载指标的方差在绝大部分采样点上都小于轮询算法、最小连接数算法的集群的带宽负载指标的 1/3，在约 20% 的采样点上三者的值相近或基于本文算法的集群的带宽负载略大。可见，本文的负载均衡机制更优。



(a) mission num=3000, rate=7 missions/s (b) mission num=3000, rate=10 missions/s

(c) mission num=3000, rate=13 missions/s (d) mission num=6000, rate=10 missions/s

图 4 不同负载均衡方法在带宽利用率方差上的对比

Fig. 4 Comparison of different load balance approaches in variance of bandwidth utilization rate

为验证在其他请求数量、请求速率下，本文算法是否同样能有效均衡负载，以 7 missions/s 和 13 missions/s 的速率发送 3000 个服务请求，以 10 missions/s 的速率发送 6000 个服务请求，重复实验 3 次，实验结果如图 4(a)、图 4(c)、图 4(d)所示。可以看出，在不同任务请求频率(missionrate)及任务数量(missionnum)的情况下，得到的结果与上一实验结果一致，即本文的负载均衡机制更优。

另外，从图 4 可以发现，当任务到达率逐渐变大时，所有算法的负载均衡指标都变大，这意味着负载均衡状况在变差，且本文算法的均衡效果同另外两种算法的均衡效果的差异在减小。其原因在于，当请求频率增大时，所有服务器的负载都趋于满负荷运行，使得不同算法的负载均衡效果的差异逐渐减小。

表 3—表 5 列出了在本实验中各算法的失败任务数、任务等待时长与松弛时长之比的均值、集群服务的能力与质量。其中 $QCoS$ 出现负值是因为计算时参数 γ 取值较大。

表 3 基于轮询算法的负载均衡结果

Table 3 Results of load balance based on round-robin algorithm

Mission num	Mission rate	Fail num	Average r_wait	QCoS
3000	7	132	0.055 21	0.609 40
	10	242	0.069 31	−1.384 25
	13	371	0.084 76	−2.482 75
6000	10	548	0.066 51	−2.747 98

表 4 基于最小连接数算法的负载均衡结果

Table 4 Results of load balance based on least connections algorithm

Mission num	Mission rate	Fail num	Average r_wait	QCoS
3000	7	117	0.053 11	1.322 36
	10	239	0.066 09	−1.193 47
	13	372	0.057 45	−1.970 40
6000	10	527	0.041 95	−2.307 35

表 5 基于本文机制的负载均衡结果

Table 5 Results of load balance based on the proposed approach

Mission num	Mission rate	Fail num	Average r_wait	QCoS
3000	7	2	0.050 37	12.962 03
	10	100	0.069 53	10.068 99
	13	217	0.081 26	6.267 65
6000	10	242	0.065 03	9.384 34

比较表 3—表 5 可知，在失败任务数占比上，当任务请求频率较低(7 missions/s)时，基于本文机制的负载均衡方法的失败任务数约占 0.07%，远低于基于轮询和基于最小连接数的负载均衡算法的 4.4%和 3.9%。当任务请求速率逐渐增大时，三者的失败任务数占比均增大。在任务请求频率为 13 missions/s 时，基于本文机制的负载均衡算法的失败任务数约占 7%，依旧低于基于轮询和基于最小连接数的负载均衡算法的 12.4%，并且三者间的相对差距在减小，原因在于请求频率增大时，各服务器都趋于满负荷运转，使得失败任务数占比增大。在等待时长与松弛时长之比的均值上，4 种不同条件下三者的值均在 5%～9%之间，且在同一条件下，三者的均值相差不大，说明了基于本文机制的负载均衡方法能够在保证低任务失败率的同时，降低计算复杂度、减少计算资源占用，以提升实时性。

综上，当任务请求频率、任务数量相同的情况下，基于本文算法的集群失败任务数更少，等待时长与松弛时长之比的均值差异不大。当综合考虑失败任务数占比、等待时长与松弛时长之比的均值、带宽负载均衡状况后，同基于轮询和基于最小连接数的负载均衡算法相比，基于本文机制的负载均衡方法能够更好地发挥服务器的集群性能，使集群拥有更高的服务质量和能力。

结束语 针对流媒体服务器集群中，任务对带宽的较高需求且不同任务类型间存在的带宽需求差异，本文提出了一种基于状态表的流媒体集群带宽负载均衡机制，通过离线计

算各种系统状态、任务状态的方式,为实际在线运行节省计算时间、提供调度策略。通过仿真实验,该算法能够用与轮询算法、最小连接数算法相差无几的调度响应时长更好地均衡服务器带宽负载状况,减少失败任务数,提升其服务质量与能力。

本文算法同样具有一定局限性,即集群空载带宽能力发生变化时,需要重新进行离线计算。同时,离线计算的调度策略无法考虑到实际运行时即将到来的任务请求,即其调度策略是当前的最佳策略,无法保证是全局最优解。此后可以在离线计算时考虑实际运行时即将到来的任务情况的因素并对其进行优化。

参 考 文 献

[1] SHU W Q. 0.3% Bandwidth Acceleration Effectively Promotes GDP Growth[J]. Communication World,2011(46):19.

[2] JIN Q. Audio and Video Data Traffic Will Account for 79% of New Traffic in 2020[N]. People's Posts and Telecommunications News,2016-05-10(6).

[3] GUL K S Q,WANG P,LUO S L,et al. A Techenical Research on High-concurrency Web Application[J]. Netinfo Security,2017(12):29-35.

[4] LIU Y,WANG L S,GUO G C. Design and Implementation on Self-adaptive Dynamic Load Balance Services in EJB Cluster System[J]. Application Research of Computers,2008(7):2064-2067.

[5] CAI Y T. Research and Application of Streaming Media Load Balancing Based on Nginx[D]. ChengDu: University of Electronic Science and Technology of China,2019.

[6] WANG Z. Research on Load Balancing Strategy of Streaming Media Server Cluster[D]. Xi'an: Xi'an University of Posts & Telecommunications,2017.

[7] LI J,NIE Y F,ZHOU S J. A Dynamic Load Balancing Algorithm Based on Consistent Hash[C]// Proceedings of 2018 2nd IEEE Advanced Information Management,Communicates,Electronic and Automation Control Conference (IMCEC). Xi'an: IEEE Press,2018:2387-2391.

[8] WEN Z,LI G,YANG G. Research and Realization of Nginx-based Dynamic Feedback Load Balancing Algorithm[C]// Proceedings of the 2018 IEEE 3rd Advanced Information Technology,Electronic and Automation Control Conference (IAEAC). Chongqing: IEEE Press,2018:2541-2546.

[9] ZHONG H,FANG Y,CUI J. LBBSRT: An Efficient SDN Load Balancing Scheme Based on Server Response Time[J]. Future Generation Computer Systems,2017,68(Mar.):183-190.

[10] GUO C C,YAN P L. A Dynamic Load-balancing Algorithm for Heterogeneous Web Server Cluster[J]. Chinese Journal of Computers,2005,28(2):179-184.

[11] GAO Z B,PAN Y C,HUA Z,et al. Improved Load Balancing Algorithm Based on Weighted Least-connections[J]. Science Technology and Engineering,2016,16(6):81-85.

[12] XU F,WANG S C,YANG W X. Could Resource Scheduling Algorithm Based on Game Theory[J]. Computer Science,2019,46(6A):295-299.

[13] WANG W B,YE Q W,ZHOU Y,et al. Dynamic Load Balancing Algorithm Based on Queuing Theory Comprehensive Index Evalution[J]. Telecommunications Science,2018,34(7):86-91.

[14] WANG Z,LIU Z Y. An Improved Dynamic Load-balancing Algorithm for StreamingMedia Cluster[J]. Computer & Digital Engineering,2018,46(2):241-246.

[15] LI G,QU W L,TIAN F,et al. An Improved Cycle Adaptive Dynamic Load Balancing Algorithm[J]. Journal of Chinese Computer Systems,2015,36(7):1476-1480.

[16] MA J Y,DING G G,WANG R Y. A New Load Balancing Method Based on Simulated Annealing Algorithm in Streaming Media System[C]// Proceedings of the 2012 8th International Confe-rence on Wireless Communications,Networking and Mobile Computing. Shanghai: IEEE Press,2012:1-4.

[17] PAN K,CHEN J Q. Load Balancing in Cloud Computing Environment Based on an Improved Particle Swarm Optimization [C]// Proceedings of the 2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS). Beijing: IEEE Press,2015:595-598.

[18] ZHENG B L,LI Y H. Study on SDN Network Load Balancing Based on IACO[J]. Computer Science,2019,46(6A):291-294.

[19] LI Y M,YAN H. Research on Cloud Computing System Resource Load Balancing[J]. Computer Measurement & Control,2016,24(10):219-221,225.

[20] SUN J W,ZHOU L,DING Q L. Research of Dynamic Load Balancing Based on Simulated Annealing Algorithm[J]. Computer Science,2013,40(5):89-92.

[21] LIU B,XU J M,DAI S H,et al. Load Balancing Algorithm Based on Linux Vitrual Server[J]. Computer Engineering,2011,37(23):279-281,287.

ZHENG Zeng-qian, born in 1995, postgraduate. His main research interests include streaming media server and load balancing.

MENG Li-min, born in 1963, Ph.D, professor, Ph.D supervisor, is a member of China Computer Federation. Her main research interests include wireless communication and network, streaming media transmission and IoT communications.