面向高并发民航业务的动态负载均衡设计

田 丰[1] 倪兆阳[2] 丁建立[2] 王 静[2]
[1](中国民航信息网络股份有限公司研发中心 北京 100000)
[2](中国民航大学计算机科学与技术学院 天津 300000)

摘 要 针对民航业务实时并发请求多、业务功能复杂、数据量大的特点,简要介绍当前业务系统的总体结构和负载均衡机制,分析现有负载均衡算法的优点和不足。为了提高民航业务系统的鲁棒性和可靠性,根据其软硬件结构特点在原负载均衡机制的基础上提出改进的动态负载均衡策略。同时设计维护负载度量指标所需的数据结构以及在这种数据结构下维护该指标的方法,并进行实验对比验证。实验结果表明,动态负载均衡策略能够更好地应对高并发复杂网络环境,使系统的鲁棒性和可靠性得到提升。

关键词 动态 负载均衡 高并发 民航 分布式

中图分类号 TP319 **文献标识码** A DOI: 10.3969/j.issn.1000-386x.2016.10.001

DYNAMIC LOAD BALANCE DESIGN FOR HIGH CONCURRENCY CIVIL AVIATION BUSINESS

Tian Feng[1] Ni Zhaoyang[2] Ding Jianli[2] Wang Jing[2]
[1](*Research and Development Center, Travel Sky Technology Limited, Beijing 100000, China*)
[2](*School of Computer Science and Technology, Civil Aviation University of China, Tianjin 300000, China*)

Abstract For the features of civil aviation business such as a lot of real-time concurrent requests, complicated business functions and large data amount, we briefly introduced the overall architecture and load balance mechanism of current business system, and analysed the merits and drawbacks of existing load balance algorithm. To promote the robustness and reliability of civil aviation business system, we put forward an improved dynamic load balance strategy based on the original mechanism in accordance with the software and hardware feature. Meanwhile we designed a new data structure required for maintaining the load measurement indicators and the relative method to maintain such indicator in this structure, and carried out experiment for contrast verification. Experimental result indicated that the dynamic load balance strategy could cope well with high concurrency and complicated network environment, which promoted the robustness and reliability of the system.

Keywords Dynamic Load balance High concurrency Civil aviation Distributed

0 引 言

随着经济社会的发展,航空运输变得越来越普及。2014 年中国民航旅客运输量已经达到 3.9 亿人次,排名世界第二。互联网的迅猛发展也使越来越多的人从网络渠道获得关于航班的各种信息。以民航旅客服务信息系统中目前为止最大规模的服务器集群 DIP(Data Integration Platform)来说,其吞吐量持续峰值已经达到 10 000 TPS,日处理消息量在 2 亿条左右。这意味着越来越多的民航业务要由信息系统处理,系统负载量大幅增加。

民航旅客服务信息系统的运行状况不仅仅关系到提供给地面用户的服务质量,还涉及到空中的交通安全和全国各航空公司的飞行计划安排,一旦出现差错将造成大量的旅客滞留和航班延误,其损失是不可估量的。这就要求民航旅客服务信息系统能够良好地应对高并发高负载环境。提高现有信息系统对高并发、动态变化、连续高可靠性的适应能力以及对其他需求的应对能力成为当前的重要任务。

民航旅客服务信息系统属于大规模分布式系统。在分布式环境下,系统中往往有多个处理能力各不相同的节点,而且请求的到达以及请求的复杂性都是随机的。如果一味地将请求全部分配给系统内某一台服务器,则不但会造成该服务器负担过重容易崩溃而且也没有充分利用系统资源[1]。单台服务器处理能力的提高是有限度的,所以如何对请求进行分流,实现灵活的负载均衡,从而达到对整个分布式系统高效利用的目的成了企业亟需解决的问题。

1 负载均衡分类与现状

负载均衡大体上可以分为静态负载均衡和动态负载均衡两大类[2]。

静态负载均衡通常以网络流和启发式算法为理论基础,依据分布式网络中的静态指标来进行请求分流。网络流法是一种重要的计算机图论方法,使用最大流最小割算法求解[3]。Stone

收稿日期:2015-07-06。民航局科技创新引导资金专项(MHRD 20130106,MHRD20140106,MHRD20150107);中国民航大学中央高校基金项目(3122014P004,3122014C016)。田丰,工程师,主研领域:中间件及云计算架构。倪兆阳,硕士生。丁建立,教授。王静,讲师。

研究了将计算开销和服务器之间的通信开销考虑在内情况下的请求分流问题，使用最大流最小割算法给出了具有两个处理器的系统中的最优分配方案，但是没有给出具有多个处理器情况下的解决方案[4]。Bokhari 研究了双处理器系统中的动态分配模型，并对 Stone 的研究进行了扩展，使其能够解决任务通信图为树形结构的请求分流问题。Towsley 又进一步研究，将 Bokhari 的成果推广到了串并联通信图结构[5]。静态负载均衡算法固然有其自身的优势，但是系统静态指标或历史经验数据不能动态反映分布式系统负载情况的变化，所以静态负载均衡不能根据实际情况动态调整负载策略[6]。

动态负载均衡算法以服务器的实时负载状态信息来决定任务的分配，其负载策略由当前系统状态决定[7]。研究者在近些年提出了许多动态负载均衡算法，按照算法的控制位置特征这些算法大致可以分为分布式、集中式和混合/层次三类。分布式负载均衡算法的基本策略就是邻近法，每一台服务器与邻近的服务器交换负载，并通过多次迭代达到全局负载均衡。比如 Kolb 等针对 MapReduce 可能产生的节点瓶颈问题提出了基于块的负载均衡算法 BlockSplit。集中式负载均衡算法中，处于逻辑上中心位置的节点收集所有其他节点的负载情况，根据全局的负载信息做出决策。比如 GreedyLB 算法在全局范围选取一个当前负载量最小的服务器，将负载分配给它。为了减小负载均衡算法的系统开销，一些算法重点研究了如何根据服务器网络的拓扑结构来构建一个层次树的问题。在此基础上将服务器划分为多个域，每一层在相邻域之间进行负载均衡，并且在层间执行负载均衡，最后达到全局负载均衡的目的，这样的负载均衡算法称为混合/层次负载均衡算法。比如 Wang 等根据 ZEUS 网络框架和云计算结构特征，提出了一种具有三级层次拓扑结构的两阶段负载均衡算法[8]。通常情况下，动态负载均衡相对于静态负载均衡能够有 30% ~40% 的性能提升[9]。

2 JCF 总体结构

2.1 整体结构

JCF(Java Core Framework) 中间件平台是民航旅客服务信息系统的重要组成部分，它运行于分布式环境下，面对着不同的硬件平台、操作系统和异构的网络环境。它通过适配服务的方式来解决 JCF 平台内部业务服务与现有企业服务总线对接的问题，以及通过企业服务总线访问 JCF 平台内部业务服务时所需要的负载均衡、资源隔离、故障隔离等需求。

JCF 平台由开发工具、运行系统、管理工具三大部分组成，为业务服务的开发、部署、运行和维护提供完整的支持。

其中运行系统的功能可以分为两个层次，如图 1 所示。

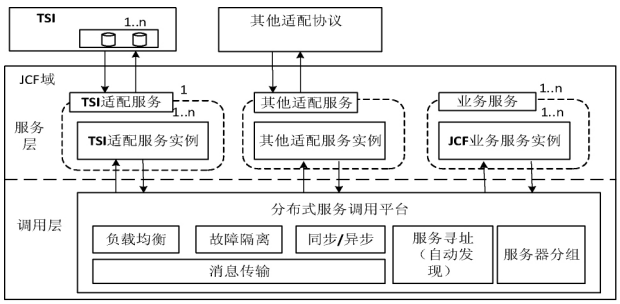

图 1 JCF 运行系统层次

服务层为应用支撑框架，允许用户使用 Java 语言创建核心业务逻辑，然后使用 JCF 提供的流程配置工具将核心业务逻辑编排成可以独立部署和运行的业务服务。平台采用 SEDA 架构，其中每个服务为一个阶段，每个阶段含有一个资源控制器，根据本阶段的负载情况动态调整资源配置。阶段之间的调用由 JCF 系统的服务调用平台负责。

调用层为分布式服务调用平台，为服务之间的相互调用提供基本支撑，包括 SEDA 异步调用、负载均衡、故障隔离、自动寻址等功能。对业务应用的开发者来说它是“透明”的。

2.2 负载均衡机制

JCF 的负载均衡机制位于服务调用方的拦截器中，作为拦截链的一个 Handler 存在，叫做 LoadBalanceHandler。LoadBalanceHandler 从属于服务调用者，调用负载均衡器 LoadBalancer 选择服务实例。LoadBalancer 是独立于拦截器 Handler 而单独存在的，是对负载均衡算法的抽象，如图 2 所示。

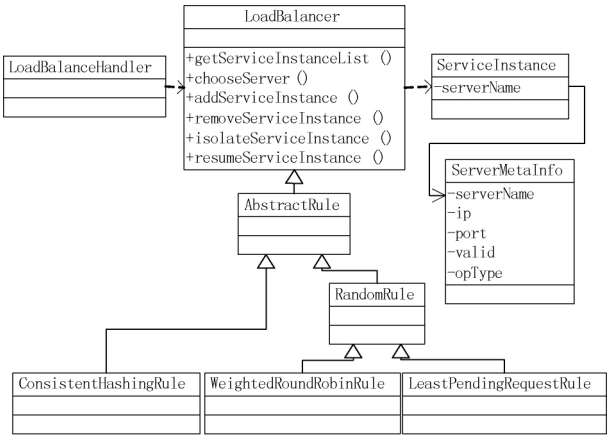

图 2 负载均衡器

每一个 LoadBalancer 实例都是当前服务器中的某一个被调用服务的负载均衡算法的抽象。其中主要的方法是 chooseServer，用来选择一个服务器处理当前请求。ServiceInstance 是对服务实例的抽象，其中封装了服务实例所在服务器的主机地址和端口。ServerMetaInfo 是用来描述服务器的对象。AbstractRule 是所有负载均衡算法的抽象基类。ConsistentHashingRule 是一致性哈希算法。WeightedRoundRobinRule、LeastPendingRequestRule 是不同策略的随机负载均衡算法。当负载均衡逻辑选择服务实例时抛出异常时，根据指定的环境变量，决定中止调用并返回错误信息，或者使用父类的随机负载均衡算法 RandomRule 随机选择一个服务实例。

负载均衡器是在客户端的服务器，由于在一台服务器上有可能存在多个服务作为某个服务的客户端，因此需要建立一个被调用服务的唯一的负载均衡器。JCF 平台采用工厂模式创建负载均衡器，LoadBalancerFactory 是负载均衡器的工厂类，便于扩展新的负载均衡策略; LoadBalanceManager 作为负载均衡器的管理者，负责保存服务器内共享的唯一的负载均衡器，如图 3 所示。

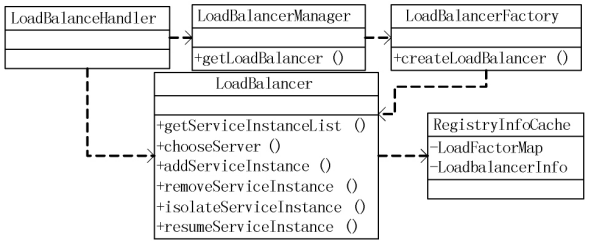

图 3 负载均衡器的管理机制

在当前 JCF 平台的固定比例因子负载均衡算法中,负载均衡的 Handler 通过 LoadBalanceManager 调用 LoadBalanceFactory 创建 LoadBalancer 的实例,并调用 chooseServer( ) 方法选择接收请求的服务器。而在 chooseServer( ) 方法中依赖服务注册库维护的 LoadFactorMap 和 LoadbalancerInfo 按照固定比例因子算法选择服务器。其中 LoadBalanceInfo 用来保存一个服务的固定比例因子的负载均衡信息,包括服务实例列表和根据比例因子计算出的最大值、最大公约数、轮转周期、轮转列表,以及表示轮转列表当前位置的的下标。LoadFactorMap 是维护服务注册库信息和 LoadBalanceInfo 的类,当服务实例增加或减少时,重新计算固定比例因子的轮转列表,并设置到 LoadBalanceInfo 中。

例如某个 JCF 域内含有 3 台 JCF 服务器,负载均衡因子分别设置为 1、2、3。假设请求处理速度远远小于请求到达速度,当有 60 个请求同时到达时这些请求将会以 3∶2∶1 的比例被分配给域内 3 台服务器。

该算法属于静态负载均衡算法,实现简单且复杂度低,符合算法设计原则。但是其比例因子只能根据机器性能和以往经验由人工设定,在运行过程中不能根据负载情况的变化实时调整策略。如果运行过程中某台服务器负载过大,该状况并不能为负载均衡算法所察觉,仍然会按照固定比例向其发送请求,而不能让其他运行状况尚良好的服务器多分担请求。所以该算法的灵活性欠佳,需要对其进行改进。

3　动态负载均衡设计

为了改善负载均衡算法的灵活性,新的算法采用动态负载均衡策略,以未完成请求数为负载度量指标,循环服务列表中的每一个服务实例,选择未完成请求数最低的服务实例处理请求。

3.1　未完成请求数的维护

国际民航系统间报文交互基于 IATA 所定义的国际规范,该规范分 TYPE-A 和 TYPE-B 两种类型,其中 TYPE-A 属于 Request-Reply 模式,TYPE-B 属于 One-way 模式。JCF 平台提供了基于这两种模式的负载均衡机制。未完成请求数的维护方式取决于消息传输模式:

1) 在 One-way 模式下,使用服务端的队列深度。当收到服务端的 Signal 信号时,将相应服务实例的指标值更新为信号中携带的请求队列深度。

2) 在 Request-Reply 模式下,使用客户端维护的指标。当负载均衡 Handler 选出服务器后,对该服务实例的未完成请求数加 1;当收到服务端的应答消息时,对相应服务实例的未完成请求数减 1;当收到超时信号时,由于同一个请求消息收到超时信号后就不会再收到应答消息,所以对相应服务实例的未完成请求数减 1。

在 Request-Reply 模式下客户端需要按照一定的时间窗口进行计数。只计算指定长度时间窗口内的请求数,在时间窗口内分为不同的刻度分别统计,每过一个刻度,就抛弃窗口外的计数,这样就完成了对超时请求数的抛弃。

3.2　未完成请求数的统计

统计过去 s 个时间单位的未完成请求数的算法,可以表述为获取当前时间 t 以前 s 个时间单位(μ) 的未完成请求数 $reqno$ 之和:

$$\sum_{n=\lfloor \frac{t}{\mu} \rfloor-(s-1)}^{\lfloor \frac{t}{\mu} \rfloor} reqno\ [n]$$

未完成请求数的统计需要使用一个适当的数据结构,然后处理好三个问题:

1) 当请求发出时,使未完成请求数加 1;

2) 当应答收到时,使未完成请求数减 1;

3) 当需要统计时,统计出指定时间范围内的未完成请求数之和。

3.3　数据结构

在计算机中模拟这一统计过程只需关心 s 个时间单位内的未完成请求数,所以给每个服务实例定义一个包含 s 个结点的线性表。每个结点代表一个时间窗口,即一个时间单位,其中保存建立该结点时的时刻 $time$ 以及一个时间单位内的未完成请求数 $reqno$ (使用原子数据类型) 。新的结点加在后面,当加入新的结点时,各结点依次前移,把时间最久远的结点抛弃,如图 4 为一个 $s = 5$ 的数据结构。

	0	1	2	3	4
time	20	21	22	23	24
reqno	3	2	5	8	7

图 4　数据结构

3.4　增加未完成请求数

假设采用结构数组方式实现该数据结构,命名为 $cell$ 。同时假设当前时刻为 t ,如果 $t - cell[s-1].time < \mu$,则 $cell[s-1].reqno = cell[s-1].reqno + 1$ 。

设 $\mu = 1$,$t = 25.74$ 时,格子的内容如下所示:

	0	1	2	3	4
time	21	22	23	24	25
reqno	2	5	8	7	1

判断条件 $25.74 - 25 < 1$ 成立,则 $cell[4].reqno = 1 + 1 = 2$。

如果 $t - cell[s-1].time \geqslant \mu$,则首先需要滑动时间窗口,再进行未完成请求数的累计。

3.5　滑动时间窗口

首先计算 t 时刻需要滑动的窗口数 n。$n = \left\lfloor \frac{(t - cell[s-1].time)}{\mu} \right\rfloor$,则需要将数据结构向右滑动 n 个窗口。例如 $\mu = 1$,$t = 27.65$ 时窗口的内容如下所示:

	0	1	2	3	4
time	21	22	23	24	25
reqno	2	5	8	7	1

计算 $n = \left\lfloor \frac{(27.65 - 25)}{1} \right\rfloor = 2$,需要向右滑动 2 个空窗口,追加后窗口的内容如下所示:

	0	1	2	3	4
time	23	24	25	26	27
reqno	8	7	1	0	0

再按照 $t - cell[s-1].time < \mu$ 的情况进行增加请求数操作,如下所示:

	0	1	2	3	4
time	23	24	25	26	27
reqno	8	7	1	0	1

3.6　减少未完成请求数

当收到应答消息时,需要减少未完成请求数。首先,从应答消息中得到请求发出的时间 t_0 ,计算出 $pos = \left\lfloor \frac{(t_0 - cell[0].time)}{\mu} \right\rfloor$ 。

如果 $pos \geqslant 0$，则 $cell[pos].reqno = cell[pos].reqno - 1$；如果 $pos < 0$，则该请求的计数已经被抛弃，那么就忽略该消息。

假设 $\mu = 1$，$t = 27$ 时窗口的内容如下所示：

	0	1	2	3	4
time	23	24	25	26	27
reqno	8	7	1	0	0

若此时收到一个应答消息，其对应的请求的发出时间 $t_0 = 24.38$，计算 $pos = \left\lfloor \frac{(24.38 - 23)}{1} \right\rfloor = 1 \geqslant 0$，则 $cell[i].reqno = cell[i].reqno - 1 = 6$，窗口内容变为：

	0	1	2	3	4
time	23	24	25	26	27
reqno	8	6	1	0	0

但假如此时收到的应答消息所对应的请求的发出时间 $t_0 = 18.65$，计算 $pos = \left\lfloor \frac{(18.65 - 23)}{1} \right\rfloor < 0$，则抛弃该应答消息。

3.7 统计未完成请求数

设当前时间为 t，如果 $t - cell[s-1].time < \mu$，那么只需要对当前的数据结构进行累积：

```
int acc = 0;
for(int i = 0; i < s; i++) {
  acc += cell[i].reqno
}
```

如果 $t - cell[s-1].time \geqslant \mu$，也就是说没有最近一个时间单位的未完成请求数，那么首先需要滑动时间窗口，然后再按上述方式进行统计。

3.8 负载均衡算法

对上述未完成请求数维护方法进行分析，发现不论是发出请求还是收到应答均需要判断是否需要滑动时间窗口。如果滑动了时间窗口，还需要对每个服务实例的总体未完成请求数进行更新以提供负载均衡操作所需要的最新参考指标。对各种操作归纳总结，提炼出公共操作后绘制出算法流程如图 5 所示。

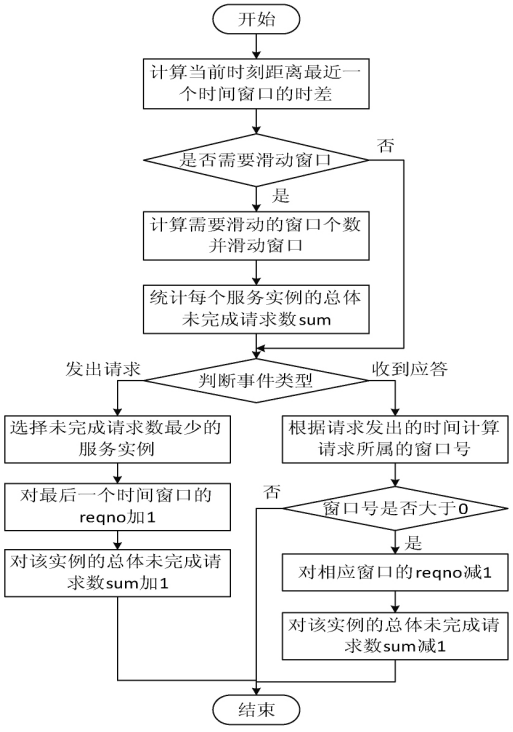

图 5 算法流程图

针对上述数据结构特点，分析负载均衡流程，设计出 Request-Reply 模式下的动态负载均衡算法：

a) 计算当前时刻 t 与最近的时间窗口所记录的时刻的差值 $\Delta t = t - cell[s-1].time$；

b) 判断 Δt 是否大于一个时间单位 μ，若是则转到步骤 c)，否则转到步骤 e)；

c) 计算需要滑动的窗口个数 $n = \left\lfloor \frac{\Delta t}{\mu} \right\rfloor$，并将窗口向右滑动 n 个单位；

d) 统计每个服务实例的未完成请求数，第 k 个服务实例的总体未完成请求数 $sum_k = \sum_{i=0}^{s-1} cell[i].reqno$；

e) 判断事件类型，若为发出服务请求则转步骤 f)，否则转步骤 h)；

f) 遍历所有服务实例的未完成请求数，选择其中的最小值 $sum_{\min}$；

g) 对第 min 个服务实例的最近一个时间窗口的未完成请求数加 1，即 $cell[s-1].reqno = cell[s-1].reqno + 1$，然后对 $sum_{\min}$ 加 1，算法结束；

h) 从应答消息中得到请求发出的时间 t_0，计算出该应答所对应的时间窗口编号 $pos = \left\lfloor \frac{(t_0 - cell[0].time)}{\mu} \right\rfloor$；

i) 判断 pos 是否大于 0，若是则转步骤 j)，否则算法结束；

j) 从应答消息中得到服务实例编号 $back$，对第 $back$ 个服务实例的第 pos 个时间窗口的未完成请求数减 1，即 $cell[pos].reqno = cell[pos].reqno - 1$，然后对 sum_{back} 进行减 1 操作，算法结束。

4 实 验

本节给出了采用动态负载均衡算法的服务器集群和采用固定比例因子负载均衡算法的服务器集群的性能对比实验结果。两个服务器集群 cluster1 和 cluster2 均部署 JCF 平台，通过适配服务接收服务总线上的请求消息。两个集群内均含有 3 台业务服务器和 1 台适配服务器，其中业务服务器上均部署有 serviceA 和 serviceB 两种业务服务，适配服务器上部署适配服务 adapterService。serviceA 用来模拟简单业务服务，不设置延迟；serviceB 用来模拟复杂业务服务，其延迟时间设置为 3 秒。如图 6 所示。

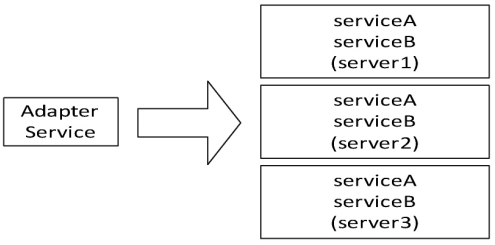

图 6 实验场景

两个集群的服务器配置相同，具体配置信息如下：

- Intel(R) Xeon(R) CPU X7550 @ 2.00 GHz;
- 8 GB RAM;
- Red Hat Enterprise Linux Server release 6.3 (Santiago);
- SUN JDK Version 1.6.0_10。

其中 cluster1 采用动态负载均衡算法，cluster2 采用固定比

例因子负载均衡算法,比例因子设置为 1∶1∶1。按照 JCF 平台的处理机制,当请求消息在服务总线上出现后,将由适配服务负责获取该消息,然后执行负载均衡算法选择集群内的一台服务器,将请求交由该服务器进行处理。

实验环境搭建完成后用客户端程序向服务总线持续发送 serviceA 的请求消息,发送间隔为 3 毫秒,该操作模拟民航请求中并发量大,同时处理速度较快的航班可售库存查询。第 30 秒钟时发送一条 serviceB 的请求消息,模拟民航请求中随时可能出现的业务功能复杂的运价搜索或航班计划创建请求。运价搜索请求的处理需要在多条航路中综合判断以返回符合搜索条件的结果,该过程会涉及到多航段排列组合进行运价计算;而航班计划创建会对原有航班计划产生连锁影响,需要进行大量计算以安排出新的计划表。这两种请求都会耗费大量服务器资源,降低服务器吞吐率。实验过程持续 90 秒钟,结束后统计出两个服务器集群的 TPS 数和集群内各台服务器的 TPS 数,截取其中 15 秒钟的数据绘制成如图 7 – 图 9 所示。

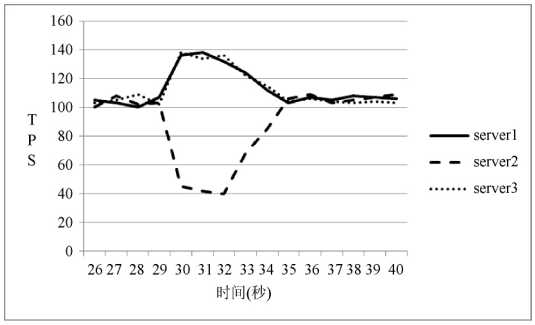

图 7　cluster1 内各服务器 TPS

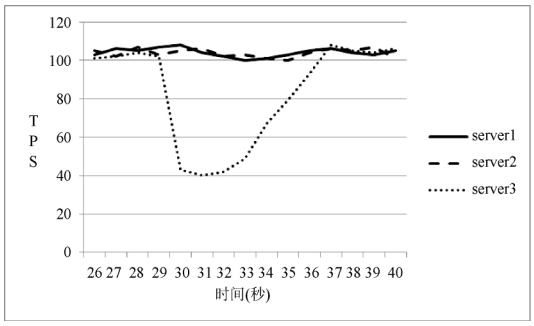

图 8　cluster2 内各服务器 TPS

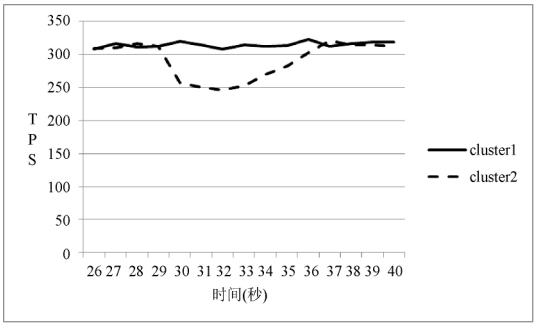

图 9　集群 TPS

从图中可以看出,在第 30 秒时两个集群中均有一台服务器的 TPS 数突然下降,这是由于请求消息中出现了对 serviceB 的请求,该请求会占用大量服务器资源,导致其 TPS 数下降。不同的是 cluster1 内另外两台服务器的 TPS 数在此种情况下出现了上升,集群 TPS 保持稳定;而 cluster2 内另外两台服务器的 TPS 数没有明显变化,集群 TPS 出现下降的情况。这说明动态负载均衡起到了应有的作用,在集群内某台服务器执行了运价搜索或航班计划创建之类的复杂请求。队列中出现积压请求消息的情况后,动态负载均衡算法根据各台服务器未完成请求数的情况自动将新接收到的请求多分发给另外两台负载较小的服务器,从而使集群 TPS 不受影响或受影响较小。而固定比例因子负载均衡算法则不论服务器负载情况变化,只能按照设定好的比例因子将请求按比例发送给各台服务器,集群 TPS 会因此受到业务复杂程度的影响。

5　结　语

在分布式系统中,负载均衡的目的是根据各台服务器的性能指标和负载情况来分配请求[10],服务器的处理能力和当前未完成请求数是负载均衡算法所要参考的重要指标[11]。未完成请求数是一个动态变化的指标,维护该参数首先需要设计一种动态度量策略,该策略要良好地适配系统结构和工作机制,以准确地反映系统中负载的变化情况。负载均衡算法是负载均衡策略的核心,在设计时要减小复杂度,避免负载均衡算法成为系统性能瓶颈[12]。

本文针对 JCF 平台的结构特点设计了动态负载均衡算法。该算法用一种窗口式的数据结构维护每个业务实例的未完成请求数,并且用滑动窗口的方式完成超时请求的自动丢弃。在此基础上以业务实例的未完成请求数为度量指标,将请求分发到负载量最小的服务器上,最大程度减少民航业务复杂度的变化对集群吞吐率的影响。这增强了 JCF 平台的鲁棒性,使 JCF 平台在高并发、复杂程度动态变化的民航业务环境下保持稳定的性能。

参　考　文　献

[1] 尤天舒. 基于 Agent 的集群负载均衡模型及其实验研究[D]. 吉林大学,2014.
[2] 王红斌. Web 服务器集群系统的自适应负载均衡调度策略研究[D]. 吉林大学,2013.
[3] 晋英豪. 异构网络中负载均衡和资源分配策略研究[D]. 中国科学技术大学,2015.
[4] 胡杨. 虚实结合框架下负载均衡服务集群的研究与实现[D]. 浙江大学,2015.
[5] 陈涛,肖侬,刘芳. 对象存储系统中自适应的元数据负载均衡机制[J]. 软件学报,2013,24(2):331-342.
[6] 弭伟. 基于 DHT 的分布式网络中负载均衡机制及其安全性的研究[D]. 北京邮电大学,2012.
[7] 吴和生. 云计算环境中多核多进程负载均衡技术的研究与应用[D]. 南京大学,2013.
[8] 杨际祥. 并行与分布式计算负载均衡问题研究[D]. 大连理工大学,2012.
[9] 周莹莲,刘甫. 服务器负载均衡技术研究[J]. 计算机与数字工程,2010,38(4):11-14,35.
[10] 尚志浩. OpenFlow 网络中服务器负载均衡的研究[D]. 兰州大学,2014.
[11] 吴宇文. 基于 OpenFlow 的网络负载均衡算法的研究与设计[D]. 华东师范大学,2014.
[12] 周松泉. 一种新的服务器集群负载均衡算法[D]. 南昌航空大学,2012.