基于网管业务特征的负载均衡策略研究

徐 昆[1], 夏玉福[2]
(1. 中国人民解放军理工大学 指挥自动化学院, 南京 210007; 2. 国防科学技术大学, 长沙 410073)

摘 要: 根据用户管理业务的组织结构提出适合网管业务特征的负载均衡策略,描述了策略的算法和实现,并在原型系统中进行测试比较,验证了算法的有效性。

关键词: 网络管理; 服务; 多流程引擎; 负载均衡
中图分类号: TP393 **文献标志码**: A **文章编号**: 1001-3695(2012)03-1054-03
doi:10.3969/j.issn.1001-3695.2012.03.069

Operation process complexity modeling and validation verification

XU Kun[1], XIA Yu-fu[2]
(1. *Institute of Command Automation, PLA University of Science & Technology, Nanjing* 210007, *China*; 2. *National University of Defense Technology, Changsha* 410073, *China*)

Abstract: This paper proposed a load balancing strategy according to the organizational structure of the user management business, described the algorithm and implementation of strategy, and verified the effectiveness of the algorithm in the prototype system.

Key words: network management; services; multi-engines of process; load balance

在面向服务体系结构的服务运行环境中,原子服务可以通过流程引擎组合成组合服务,并将这个组合流程持久化存储到服务能力库中。在大量用户业务的环境中,单流程引擎可能出现负载过重导致无法满足对响应时间的要求,为提高资源利用效率和缩小系统瓶颈,需要配置多流程引擎环境。网络管理服务活动中,多项用户业务同时部署或者部署一项用户业务可能需要多个组合服务,尤其是需要大规模并行服务和多个组合服务同时运行的情况下,流程引擎的负载均衡策略十分重要。

由于服务之间存在顺序和依赖关系,因此服务的负载均衡问题具有一些特殊性,国内外对服务路径的负载均衡问题进行了很多研究[1,2]。文献[3]提出了一种通过特定中间节点构建路径的算法,从初始覆盖网络拓扑图派生出一个多层转换图,使用 Dijkstra 最短路径算法,求出经过某些特定节点的最短路径。文献[4]提出了利用状态转换图来选择符合网络带宽约束的服务路径的方法,文献[5]在此基础上提出了使用 LIAC (least-inverse-available-capacity)测度来构建服务路径的方法。文献[6]针对 LIAC 算法存储空间和计算时间开销较大、网络拓扑和负载信息难以预测的缺点,提出一种分布式的自适应服务组合负载均衡算法——基于负载容率的负载均衡算法。文献[7]提出了一种启发式模糊算法,将传统控制理论中的模糊控制和启发式策略应用到了工作流负载均衡中,把启发式规则与动态反馈负载均衡有效结合,不仅关心服务节点本身的负载信息,而且关心客户请求可能会给服务节点带来的负载即请求负载,请求派遣器和负载均衡器根据请求负载选择一个最合适的服务来处理这个请求。文献[8]针对网格环境下的负载不均衡问题,提出了一种分层动态负载均衡机制,并分析了分层负载均衡机制的有效性。文献[9]通过主动测量各探测器的负载作为预测依据,利用预测的负载值作为负载均衡的根据,提出了一种基于预测的并行入侵检测系统的负载均衡方案。

上述各类算法从节点和链路入手实现了负载均衡的目的,但没有考虑任务的运行特征,所以不适合网络管理用户业务的负载均衡策略。例如,用户紧急需要自治域的网络态势,包括节点信息、链路状态和历史记录,假设管理节点 A 正在访问该自治域,那么将此任务委托给哪个节点更为合适,如果由指定管理节点负责收集,会有多种情况:a)管理节点即是节点 A,此时可以及时完成任务;b)管理节点不是正在访问的节点 A,但是可以及时打断节点 A,此时只需付出很小的信息交流开销,同样可以迅速完成任务;c)管理节点忙或者故障,由其负责收集不如节点 A 快,此时节点信息和链路状态可以和自治域内网络单元进行信息交流获得,而历史记录却很难从网络单元中获得真实信息,并且信息交流开销很难估计,后两种情况是传统负载均衡忽略的方面。本文通过对网管现有业务构成的分析,提出了一种简单、高效的负载均衡策略,为面向服务的网络管理系统提供了有效的支撑。

1 过程分析

用户业务定制完成后交付业务管理模块进行业务部署,此时分布式管理服务运行环境执行资源调配,一项业务部署的资源调配过程如图 1 所示。

引入用户业务深度和多级流程引擎的概念:流程引擎的配置在空间上是分布式的,但在一项业务的组合过程中,区分流程引擎的级别非常重要。需要指出的是,流程引擎本身之间没有区别,而是在业务的设计过程中,明确用户业务的深度,即从

收稿日期: 2011-06-16; **修回日期**: 2011-07-28

作者简介:徐昆(1982-),男,山东济南人,工程师,博士研究生,主要研究方向为通信与信息系统、网络管理(xukunpost@163.com);夏玉福(1982-),男,硕士,主要研究方向为网络工程、网络管理.

原子服务算起,经过组合服务与子用户业务,最终形成用户业务需要多少级。定制的用户业务有如图 2 所示的几种可能。

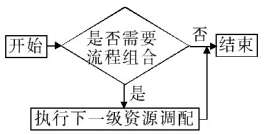

图 1　业务部署过程

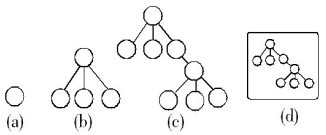

图 2　用户业务的多种形式

图 2 中几种形式可由以下四种形式解释:

a)由一个原子服务形成的用户业务,这种情况相对简单,但是业务的数目可能比较大。

b)由一个组合服务构成的用户业务,它由多个原子服务组成,但是只有一级流程引擎,也就是只有组合服务和原子服务两级资源,没有多级资源。

c)由一个组合服务构成的用户业务,但是需要下一级组合服务与其他服务组成,这时需要多级流程引擎。

d)申请的业务是其他用户已经正在运行的业务,可能是上述任何一种情况,不同的是业务正在运行中,只需要将它提交给不同的用户。

2　静态模型

根据上面分析的业务流程,初始模型构建用户业务的静态模型,将用户业务看成一个有向连通图。一个复杂的用户业务中复用原子服务或者组合任务是非常可能的,因此,如果把它看成无向图,将会有回路。下面是将有向连通图转换为赋权二叉树的过程。

a)把有向图转换成无向图,如图 3 所示。

b)把无向图转换成树,如图 4 所示。

c)将 n 元有序树转换成二叉树。找出儿子多于 2 的节点,2 个儿子一组,在不破坏儿子排序的前提下,优先将同一网段地址的节点分为一组;每组增加父节点,如果是奇数个儿子,剩余一个儿子不动;重复这一步骤,直到每个节点最多只有两个儿子,如图 5 所示。

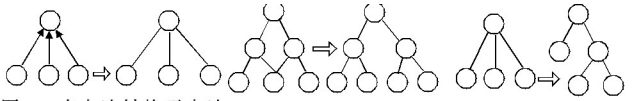
图 3　有向边转换无向边　　图 4　回路消除　　图 5　转换成二叉树

经过转换后的静态模型业务深度会增加,产生虚假的组合实例,但并没有执行时间,所以将它的执行时间设置成远远小于一般组合实例的执行时间,使其不影响组合实例对流程引擎服务器的分配。

3　动态参数

在建立起用户业务的静态模型后,需要对流程引擎节点的动态参数进行定义,第 m 个执行组合服务过程的流程引擎节点设置一个包含组合消息队列的调度器和以下参数:

a)当前未执行完的组合实例集合 $U_m = \{u_1, u_2, u_3, \cdots, u_p\}$,当前未执行完实例数 P_m,需要的时间是 $T_m = T_{u_1} + T_{u_2} + \cdots + T_{u_p}$。

b)新定制的用户业务按静态模型转换为完全二叉树后,得到所有的父节点集合 $F = \{f_1, f_2, f_3, \cdots, f_q\}$,需要的时间是 $T_F = T_{f_1} + T_{f_2} + \cdots + T_{f_q}$。

c)系统内流程引擎服务器有 n 个,那么平均每个流程引擎节点需要的执行时间 $T_a = (\sum T_m + T_F)/n$。

d)每个流程引擎服务器允许最多的组合实例数 $P_a = \lambda|1 + (\sum P_m + q)/n|(\lambda \geqslant 1)$,为第 m 个节点设置组合实例窗口 $W_m = P_a - P_m$。

4　算法描述

根据以上建立的静态模型和设置的动态参数,通过负载窗口作为调节节点负载的参考条件,需要一个全局调度器和每个流程引擎节点调度器协作完成负载的调度。管理者环境掌握着各种服务器的运行情况,可以很方便地收集全局调度器所需的各个参数,因此在管理环境中设置全局调度器,仅负责业务组合实例的分配。

1)全局调度器　执行的用户业务首先要经过全局调度器的转换和分配。全局调度器存储并实时更新各个流程引擎服务器的地址集合 A、所有流程引擎服务器节点组合实例集合 P 与执行时间集合 T'。全局调度器调度算法如下:

```
输入:用户业务。
输出:业务实例的节点分配。
if 当前接收新用户业务 then
{遍历每个业务实例节点,对每个有多个父节点的节点
{复制节点 N-1 次,N 等于父节点数;
 将每个复制的节点标记深度最大的节点名;}
 遍历每个业务实例节点,对每个多于 2 个儿子的节点
{
a)if 奇数儿子
{每 2 个儿子组成一对,同一地址优先组合,并增加父节点,设置执行时间为 C;//C 为常数
增加的父节点为偶数并且大于 2,重复步骤 a);
奇数进入步骤 b);}
b)if 偶数儿子
{每 2 个儿子组成一对,同一地址优先组合,并增加父节点,设置执行时间为 C;//C 为常数
增加的父节点为偶数并且大于 2,重复步骤 b);
奇数进入步骤 a);}
}
计算 Ta、Pa; //λ 暂时设置为 1
按业务深度降序遍历每个组合实例节点,在二叉树中就是每个父节点,对每个节点
{
在同一网段地址中按负载升序寻找流程引擎节点
if(匹配成功)
{
执行 instanceTo(S,D, Ta, Pa,I);
修改执行此实例的流程引擎节点状态参数;
}
else
{直接按负载升序分配组合实例执行 instanceTo(S,D, Ta, Pa,I);
修改执行此实例的流程引擎节点状态参数;}
}}
```

2)流程引擎节点调度器　流程引擎节点对全局调度器的调配持乐观态度,即不拒绝来自全局调度器分配的组合实例,并且只在增加组合实例时才对负载窗口进行修改,结束实例并不改变当前负载窗口。流程引擎节点调度算法如下:

```
输入:实例事件。
输出:执行实例的节点和状态变化。
if(有到达实例事件)
{
if(Ta, Pa 为空) //非全局调度器分配的组合实例
{
T = Tm + t;//t 为组合实例执行时间
if(T < Ta)//预执行时间小于平均执行时间
{
W = Wm - 1;//负载窗口减 1
```

```
}
else
{W=0;}//负载窗口设0
if(W<1)
{向全局调度器获取剩余时间最小节点;
  执行 instanceTo(S,D, null, null,I);}
else
{接收组合实例;
Tm=T;//将预判断参数写实
Wm=W;
Pm= Pm+1;}
}
else//全局调度器分配实例
{接收组合实例;
Tm=Tm+1;//修改参数
Wm=Wm-1;
Pm=Pm+1;}
}
if(有实例结束事件)
{Tm=Tm-t;//t 为组合实例执行时间
Pm=Pm-1;
将 Pm、Tm 更新至全局调度器第 m 个节点的组合实例数及执行时间;}}
```

5　测试比较

业务访问测试中,联合拓扑管理业务是由两个原子服务组合形成新的服务后呈现给用户的业务,只有一个组合实例,从流程引擎服务器控制台中获得流程引擎。

执行组合过程的时间在 10 000 ms 左右,以这一实例为基础,设计深度为 6 的用户业务,硬件模拟环境的流程引擎服务器最多设置 5 个,而 6 级深度的用户业务所包含的组合实例数的最小值将超过 4,并且从 2 级节点开始的根节点每增加一级就使用线程在执行过程中睡眠 5 000 ms,以便于节点在队列中排序。图 6 是业务的初始有向图。

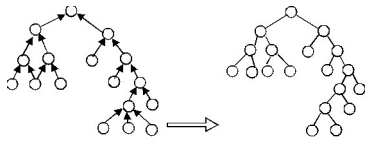

图 6　业务组合实例

将图 6 中所有有向图按照静态模型转换为二叉树的转换过程中,产生的组合过程的执行时间的常数均设置为 100 ms,这样使其在组合实例排序中处于队列的尾部,按照负载策略将会被分配到流程服务器队列的尾部,也就是负载较重的流程引擎服务器。原图转换后的二叉树高为 6,增加的虚拟组合实例为 1,忽略通信和计算的时间开销,从理论上估算此业务在单流程引擎环境的执行时间是 125 100 ms,而在多流程引擎并行执行的最理想情况下的执行时间是 25 000 ms。业务在使用负载均衡策略和使用轮询调度两种情况下分别加载得到的数据如图 7 所示。两种方法分别加载 7 次,使用均衡策略的均值是 25 525 ms,未使用均衡策略的均值是 29 664 ms;使用均衡策略的响应时间在均值附近,而未使用均衡策略的抖动较大,虽然没有出现最坏状况,但是平均耗时之差比优于均衡策略的两个结果要大得多,因此可以看出,使用负载均和策略的用户业务响应时间比较稳定。

图 8 所示是单次测试 CPU 占用率和占用时间的乘积。7 次结果的平均值中,尽管两种方法的 CPU 和内存占有率的峰值相差不是很大,但是测试中未使用负载调度的业务响应时间较长,使得 CPU 和内存占有率的峰值持续时间也较长,因此实际负载也要大很多。

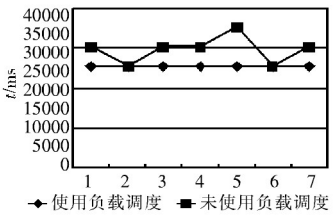

图 7　两种策略的业务响应时间曲线

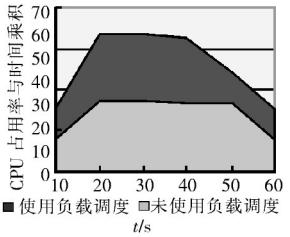

图 8　单次测试 CPU 占用率的乘积表示

测试中使用负载均衡策略和未使用的均值之差为 4 139 ms。类似地,分别加载 8 个不同的业务,得到它们的均值之差是 1 673、2 364、127、4 672、5 322、4 313、8 432 和 4 140 ms,这样一共产生 9 个均值差。9 个观察值来自不同的业务,是相互独立的,而且观察值都产生于负载策略导致的均值之差,因此,可假设它们服从正态总体分布,正态均值和方差未知,需要检验的是正态分布总体的均值 u 是否大于 0。

检验接受:$H_0>0$,拒绝域:$H_0\leqslant 0$;从得到观察值可以计算样本均值 $d=3\ 909$,样本方差 $S=2\ 376$;按单个正态总体均值的 t 检验(检验水平 0.005)的拒绝域:$t=d/(S/n_1/2)\leqslant 3.355\ 4$,计算得 $t=4.936$;t 的值未落在拒绝域内,故接受 $H_0>0$,即认为使用负载均衡策略在加载单业务且业务组合实例不少于流程引擎服务器条件下的稳定性和响应时间优于未使用的情况。

6　结束语

本文在多流程引擎环境中讨论了基于网络管理业务特征的流程引擎的负载均衡策略,并且在服务运行环境的原型系统中对多流程引擎环境中的负载均衡策略进行了测试,从实践上检验了负载均衡策略的可行性。

参考文献:

[1] SHAN Zhi-guang, LIN Chuang, MARINESCU D C, *et al*. Modeling and performance analysis of QoS-aware load balancing of Web-server clusters[J]. Computer Networks, 2002, 40(2): 235-256.

[2] GUO Cheng-cheng, YAN Pu-liu. A dynamic load-balancing algorithm for heterogeneous Web server cluster[J]. Chinese Journal of Computers, 2005, 28(2): 179-184.

[3] MA Q, STEENKISTE P. On path selection for traffic with bandwidth guarantees [C]//Proc of International Conference on Network Protocols. [S.l.]: IEEE Computer Society, 1997: 191-202.

[4] RAMAN B, KATZ R H. Load balancing and stability issues in algorithms for service composition [C]//Proc of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies. 2003: 1477-1487.

[5] TEL G. Introduction to distributed algorithms [M]. 2nd ed. Beijing: China Machine Press, 2004: 61-92.

[6] 李文中,郭胜,许平,等.服务组合中一种自适应的负载均衡算法[J].软件学报,2006,17(5):1068-1077.

[7] 戴毅,曹健.基于模糊控制的启发式工作流引擎负载均衡策略[J].通信学报,2006,27(11):284-289.

[8] 胡志刚,张艳平.基于目标约束的分层动态负载均衡算法[J].计算机应用研究,2011,28(3):1105-1107.

[9] 陈宇,梁刚,李涛.网络入侵检测系统的负载均衡方案[J].计算机工程与应用,2011,47(7).:117-119,142.