计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(1):127−131 [doi: 10.15888/j.cnki.csa.006142]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 OpenStack 的 Swift 负载均衡算法①

徐　敏[1], 李　明[1], 郑建忠[2], 孙　强[1], 管建超[1], 罗华永[2], 张　辉[2]

[1](国网安徽省电力公司信息通信分公司, 合肥 230061)
[2](北京中电普华信息技术有限公司, 北京 100192)

摘　要: 为了解决由于 OpenStack 的负载分发不均衡而引发的存储性能下降、资源利用率降低、I/O 响应时长增加等问题, 提出对加权最小连接调度算法进行改进. 通过对对象存储的负载均衡调度算法研究, 利用存储节点的 CPU、内存、硬盘、I/O 资源利用率信息, 并结合节点任务请求连接数, 计算存储节点负载能力、性能和权值. 负载均衡器根据每个存储节点的权值大小判断任务分发方向. 经实验证明改进的负载均衡调度算法能够解决存储读写性能下降的问题, 提升数据吞吐率、存储读写性能和系统稳定性.

关键词: OpenStack; 云存储; swift; 负载均衡; 加权最小连接调度算法

引用格式: 徐敏,李明,郑建忠,孙强,管建超,罗华永,张辉.基于 OpenStack 的 Swift 负载均衡算法.计算机系统应用,2018,27(1):127–131. http://www.c-s-a.org.cn/1003-3254/6142.html

Swift Load Balancing Algorithm Based on OpenStack

XU Min[1], LI Ming[1], ZHENG Jian-Zhong[2], SUN Qiang[1], GUAN Jian-Chao[1], LUO Hua-Yong[2], ZHANG Hui[2]

[1](Information and Communication Branch, State Grid Jiangsu Electric Power Company, Hefei 230061, China)
[2](Beijing China Power Information Technology Co. Ltd., Beijing 100192, China)

Abstract: In order to solve the problems of reduced storage performance, reduced resource utilization, increased I/O response, which are caused by unbalanced load distribution of OpenStack, the weighted least-connection scheduling algorithm is improved. Researching load balancing scheduling algorithm for object storage, using CPU memory disk and I/O of the storage nodes information of resource utilization, considering the number of request connections, we calculate the storage node load capacity performance and weight. Load balancing determines task distribution direction according to the size of every weight. Experiments show that the improved load balancing scheduling algorithm can optimize the performance of storage, improve the data throughput storage performance and system stability.

Key words: OpenStack; cloud storage; swift; load balancing; weighted least-connection scheduling

随着互联网技术的不断发展, 海量的视频、图像、电子商务、购物、游戏等平台数据, 现有的存储模式和存储格局已经满足不了现状需求. 对云计算的存储能力和网络带宽也提出了更高的要求. 如何快速存储大量数据、合理分发数据、动态平衡存储节点负载以及有效管理和利用数据, 对云存储技术提出了巨大的挑战. 因此, 当前的云存储技术越来越受到更多人的关注、更多人追捧, 造就云存储技术成为热点.

由于在现实的集群存储生产环境中, 经常出现一部分节点负载较轻甚至节点空运行, 而另一部分节点的负载却远远超出负荷, 从而引发负载分布不均衡问题. 由于负载不均衡问题会导致系统 I/O 读写性能下降, 增加系统 I/O 响应时长, 降低系统资源利用率, 导致系统性能得不到充分发挥[1]. 存储读写性能、存储效率、响应时长等问题屡屡暴露出来, 而负载均衡技术正是解决这类问题的关键所在[2-4].

① 收稿时间: 2017-04-04; 修改时间: 2017-04-20; 采用时间: 2017-04-27; csa 在线出版时间: 2017-12-22

© 中国科学院软件研究所 http://www.c-s-a.org.cn

近几年一些研究者提出在云环境下的负载均衡算法. 文献[5]提出一种基于人工蜂群的负载均衡算法, 选择最佳虚拟机评估其负载状态, 然后将任务分发给该虚拟机, 如果该虚拟机负载状态依然最佳则接受分配任务, 否则选择下一个级别虚拟机; 文献[6]提出一种基于蜜蜂行为的负载均衡算法 (HBB-LB), 对任务按照优先级划分对匹配虚拟机进行分配任务. 优点响应快、增加吞吐量, 但缺点是任务优先级的界定不准确; 文献[7]提出限制重定向率算法, 限制向低负载节点转发任务; 文献[8]提出周期性采集负载信息算法, 将节点分为低、中、重三类负载, 缺点是频繁分类造成延时; 文献[9]提出基于粗糙集的负载均衡算法, 将节点按照负载承载情况划分为正域、负域和边界三类.

本文在前人研究的基础上结合 swift 的复本特性, 提出一种改进的加权最小连接调度负载均衡算法解决负载不均衡导致系统 I/O 读写性能下降的问题. 利用集群性能优势[10], 即改进负载均衡资源调度算法, 通过采集存储节点的 CPU、内存、硬盘、I/O 资源利用率等信息, 计算存储节点的负载能力, 评估集群中存储节点负载情况, 合理调整任务分发数, 平衡节点负载, 从而解决了存储节点负载分发数不均衡的问题, 有效地提升了存储 I/O 读写性能和数据吞吐率.

1 算法改进

1.1 算法改进分析及思想

选择加权最小连接调度算法理由: 首先提供存储节点实时负载状态和请求任务之间的依赖关系; 其次在云环境中提供实时负载状态横向扩展的能力[11]. 尽管加权最小连接调度算法的负载均衡效果比较理想, 但其也存在以下两点缺陷.

(1) 集群中存储节点的权值存在灵活性差的缺陷, 设置不够灵活、不能准确反映各节点对负载的实时处理能力. 每个节点权值是管理员根据以往经验手动设置的. 在负载随着节点接收到的任务数量的增加而增加的同时, 其权值也会出现较大的偏差, 因此也应该对权值进行相应调整, 但手动调整权值又很繁琐, 存在不能动态调整权值缺陷. 不合理的权值分配将会影响负载均衡的效率和整个集群系统的性能[12].

(2) 加权最小连接调度算法虽然通过任务连接数表示节点负载情况, 但并不能准确反应当前任务负载的具体情况, 比如读与写的任务不同, 其处理能力也不同, 就不能准确反应节点的具体负载情况. 节点的负载情况还应该由节点的 CPU、内存、硬盘、I/O 等性能参数反映. 由于每个存储节点的处理能力和网络带宽性能差异比较大, 因此仅通过节点的任务连接数量判断节点的负载情况有点片面, 要准确判断节点的负载情况还需要参考节点的 CPU 利用率、内存性能、硬盘大小和 I/O 性能等参数[13].

综合上述分析, 在原有加权最小连接调度算法的基础上, 提出一种改进的加权最小连接调度算法, 即增加 CPU 利用率、内存性能、硬盘大小和 I/O 吞吐率四个参数. CPU 利用率、内存性能、硬盘大小和 I/O 性能也反映了节点的任务处理能力[14], 如果将 CPU 利用率、内存性能、硬盘大小和 I/O 性能与任务连接数结合应用在改进的算法中, 能够更好地反映节点的具体负载情况.

算法思想如下: 负载均衡器每隔一个周期向代理服务器节点发送一个 CPU、内存、硬盘、I/O 利用率的信息采集请求, 代理服务器节点收到该任务请求后, 会及时反馈当前存储节点的 CPU、内存、硬盘、I/O 的利用率信息. 负载均衡器会根据每个节点的 CPU、内存、硬盘、I/O 的利用率以及节点的任务连接数计算出每个节点的负载情况, 评估集群中存储节点负载能力, 合理调整任务分发数, 平衡节点负载, 从而解决了存储节点负载分发数不均衡的问题, 有效地提升了存储性能和数据吞吐率. 如果节点 CPU、内存、硬盘、I/O 性能数据采样周期值设置太短, 会增加资源开销过大[15], 根据以往的经验值建议采样周期值设置在 5-30 秒为佳[16].

1.2 算法改进设计

本文对加权最小连接调度算法改进设计分四步: 第一, 通过命令分别采集存储节点的 CPU、内存、硬盘、I/O 四个参数在空载时的最大可用资源能力和在负载时的资源利用率等信息, 并建立空载资源性能模型和负载资源模型; 第二, 为每个模型参数分配相应的权重, 权重优先级为: CPU>内存>硬盘>I/O, 且权重之和为 1; 第三, 分别对每个模型参数加权求和, 分别得出每个存储节点的性能和负载能力; 第四, 计算每个存储节点的负载权值, 即负载能力除以性能. 具体设计如下所示.

(1) 建立模型

根据表 1 设计每个存储节点在空载时的性能变量矩阵$F = [F_c, F_m, F_d, F_I/O]$.

根据表 2 设计每个存储节点在负载时的资源利用变量矩阵$L = [Lc, Lm, Ld, LI/O]$.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

表 1 空载资源模型

性能名称	CPU	内存	硬盘	I/O
性能参数	F_c	F_m	F_d	$F_{I/O}$
说明	CPU频率	内存大小	硬盘大小	I/O吞吐率

表 2 负载资源模型

负载名称	CPU	内存	硬盘	I/O
负载参数	L_c	L_m	L_d	$L_{I/O}$
说明	CPU利用率	内存使用率	硬盘使用量	I/O吞吐率

(2) 性能计算

设计一组存储节点为$S=\{S_1,S_2,...,S_n\}$, S_i 代表第 i 个节点; 设计 CPU、内存、硬盘、I/O 四个参数在存储节点空载时的最大可用资源能力的权重为 p_i, $\sum_{i=1}^{4} p_i=1$; 设计存储节点空载时的最大可用资源性能$M(S_i)$, 则$M(S_i)$的计算公式 (1) 为:

$$M(S_i)=p_1*F_c(S_i)+p_2*F_m(S_i)+p_3*F_d(S_i)+p_4*F_I/O(S_i) \quad (1)$$

(3) 负载能力计算

设计 CPU、内存、硬盘、I/O 等参数在存储节点负载时的负载能力的权重为 q_i, $\sum_{i=1}^{4} q_i=1$; 设计存储节点负载能力$N(S_i)$, 则$N(S_i)$的计算公式 (2):

$$N(S_i)=q_1*L_c(S_i)+q_2*L_m(S_i)+q_3*L_d(S_i)+q_4*L_I/O(S_i) \quad (2)$$

(4) 负载权值计算

设计每个存储节点的负载权值为$K(S_i)$, 则$K(S_i)$的计算公式 (3) 为:

$$K(S_i)=\frac{N(S_i)}{M(S_i)} \quad (3)$$

负载权值$K(S_i)$意味着存储节点的负载能力大小, 如果$K(S_i)$值大则说明该节点正处于负荷较重状态; 如果$K(S_i)$值小则说明该节点正处于负荷较轻的状态, 此时负载均衡器应该将任务分发给该节点.

1.3 算法实现

实现改进的加权最小连接调度算法主要是利用计算节点通过调用 linux 命令令获取 CPU、内存、磁盘的利用率和 I/O 吞吐率等信息, 并将信息更新到存储节点的模型表 1 和表 2, 具体方法如下.

通过 cat /proc/stat 命令获取 CPU 利用率; 通过 cat /proc/meminfo 或 free 命令获取内存利用率; 通过 time dd if=/dev/sda1 of=/temp/test.dbf bs=64k 命令获取 I/O 吞吐率; 通过 df -h 命令获取磁盘利用率.

存储节点再调用 getresource() 函数访问模型表 1 和表 2 分别采集每个存储节点的 CPU、内存、硬盘、I/O 四个参数在空载时的性能资源变量 F 和负载时的资源变量 L. 然后根据公式 (1) 计算每个存储节点空载性能, 同理根据公式 (2) 计算每个存储节点的负载能力, 存储节点再由公式 (3) 计算每个存储节点权值 $K(S_i)$. 负载均衡器根据每个存储节点的权值大小判断任务分发方向, 如果$K(S_i)$值大则说明该节点正处于负荷较重状态; 如果$K(S_i)$值小则说明该节点正处于负荷较轻的状态, 此时负载均衡器应该将任务分发给较轻节点. 实现改进后的加权最小连接调度算法的实现流程以及关键伪代码如下所示:

步骤 1. OpenStack 计算节点 nove 读取每个存储节点的资源使用信息, 包括 CPU、内存、磁盘的利用率和 I/O 的吞吐率;

步骤 2. 分别调用每个存储节点在空载情况的模型表 1 和负载情况的模型表 2, 并将步骤 1 读取到的信息填写到空载模型表 1 和负载模型表 2 中;

步骤 3. 存储节点根据空载模型表和负载模型表数据计算每个存储节点的负载能力、性能和权值;

步骤 4. 负载均衡器根据步骤 3 计算结果评估每个存储节点的负载情况, 并对负载按升序方式排序;

步骤 5. 负载均衡器侦听是否有新的任务请求, 并对新的任务占用资源进行评估, 然后调用改进的算法将任务分发给负载最小的节点处理;

步骤 6. 待存储节点处理完新的任务后, 再评估该存储节点是否超载, 如果超载转到步骤 4; 如果未超载则结束本次流程.

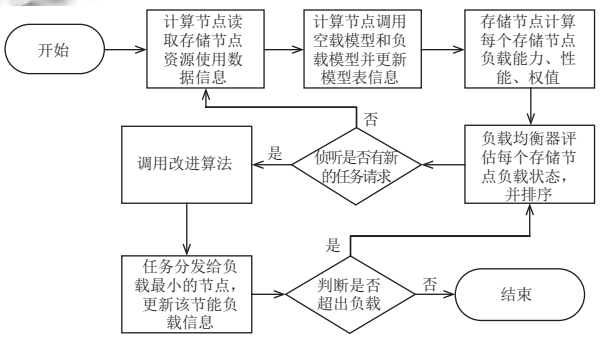

图 1 改进的加权最小连接调度算法流程图

static void getresource(information *ptr);//采集资源使用率, 资源参数在结构体中定义

static void send(information *ptr);//将采集的数据发给负载均衡器

© 中国科学院软件研究所 http://www.c-s-a.org.cn

```
static void receive(information *ptr);//负载均衡器接收资源使用率
static void update();//更新资源使用率数据
typedef struct information //定义结构体
{
  double cpuinfo;//CPU 使用率
  double meminfo;//内存使用率
  double used_disk;//硬盘使用率
  double re_disk;//硬盘读速度
  double wr_disk;//硬盘写速度
}information
double N(),M(),K();//定义负载性能、 能力、 权值
double K()=N()/M();//计算权值
```

伪代码说明: 存储节点定期调用 getresource 函数采集自身的资源使用率数据, 如 CPU、内存、硬盘和 I/O 信息, 然后通过 send() 函数将采集到的数据发送给负载均衡器; 负载均衡器再调用 receive() 函数接收存储节点的资源信息, 并调用 update() 函数更新数据. 存储节点调用负载性能函数和负载能力函数计算权值.

根据算法实现流程计算该算法空间复杂度和时间复杂度. 在该算法中每次迭代空载模型表和负载模型表空间主要用来存放存储节点的 CPU、内存、磁盘和 I/O 使用率信息, 由于存储节点数量不多且每个存储节点存储两张模型表, 每张表存储的四个参数的使用率数据不大, 模型表中的记录数量也不多, 因此, 改进后算法的空间复杂度为 O(n), n 为空载模型表和负载模型表空间, 由于两张模型表只存放四个参数的资源使用信息, 且数据量少, 相对于存储设备而言, 其占用的存储空间可以忽略不计, 故改进后算法的空间复杂度不大, 不会产生较大存储空间浪费现象. 由于实现该算法流程较简单, 且遍历每个存储节点获取 CPU、内存、磁盘和 I/O 四个参数资源使用信息数据小、执行速度快, 四个参数数量相对小于存储节点数量 n, 故四个参数时间可以忽略不计, 因此, 该算法的时间复杂度为 O(n), n 为存储节点数量, 由于存储节点数量不多, 故改进后算法的时间复杂度不大, 不会造成过多的时间开销.

改进后的算法优化了原有算法的缺陷, 解决了云环境下 swift 存储负载不均衡的问题, 有效地提升了存储 I/O 读写性能和数据吞吐率, 提升资源利用率和任务响应时间缩短最小化进程间的通信开销[17], 从而达到提高整体性能效果.

2 实验

2.1 实验环境搭建

为了验证本文改进的负载均衡算法, 搭建以下实验环境进行集群环境的存储性能测试, 以验证改进后的算法有效性. Swift 负载均衡测试环境由 4 个压力客户端、1 个负载均衡器、2 个 Proxy 服务器以及 4 个存储服务器组成, 存储单元为 SATA 磁盘以及 Intel 的万兆网卡. 压力测试工具为 COSBench.

2.1.1 部署图

负载均衡实验环境部署图如图 2 所示.

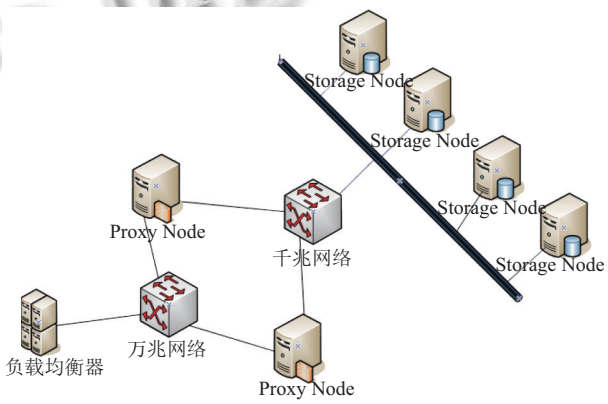

图 2 swift 负载均衡实验环境部署图

2.1.2 硬件配置

实验环境硬件配置如表 3 所示.

表 3 服务器配置

名称	数量	配置
控制节点	3	CPU2 x 8 Core, 128 GB内存, 硬盘300 G *2 SATA
计算节点	4	CPU 8 Core, 16 GB内存, 硬盘300 G *2 SATA
Swift代理	2	CPU 8 Core, 64 GB内存, 硬盘300 G *2 SATA
Swift存储	4	CPU 8 Core, 16 GB内存, 硬盘300 G *2 SATA , 22*900 GB SAS
负载均衡器	1	CPU2 x 8 Core, 128 GB内存, 硬盘300 G *2 SATA

2.2 实验结果

本文算法与加权最小连接调度算法共进行 8 组实验测试, 分别测试读小文件、读大文件、写小文件、写大文件不同任务, 并统计、记录在执行相同的任务条件下采用不同的算法读写速度, 为了提高实验的准确性, 每项实验测试两遍. 在每组实验传输大小相同文件的条件下, 共采集 2 组测试数据, 并对比算法改进前后的存储节点读写速度, 分析算法改进后的读写性能. 算法改进前后的存储读写速度对比见表 4.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

表 4 算法改进前后的存储节点读写速度对比

读小文件(MB/s)			读大文件(MB/s)			写小文件(MB/s)			写大文件(MB/s)		
	改进前	改进后		改进前	改进后		改进前	改进后		改进前	改进后
第一组	1518	1965	第三组	868	1173	第五组	699	1160	第七组	436	690
第二组	1472	1811	第四组	902	1105	第六组	653	1092	第八组	449	718

从表 4 可以看出, 本文算法在存储节点读写数据时其速度明显高于加权最小连接调度算法, 经验证改进后的算法存储节点读写速度比改进前的读写速度有明显提升, 能较好的地解决加权最小连接调度算法读写性能低、响应速度慢的问题.

当任务连接数比较小时本算法优势不明显, 但任务连接数超过 500 时本算法优势明显高于其它算法, 本算法能够保障负载均衡趋于平衡, 而其它算法促使负载失衡. 根本原因本算法具有权值动态调整功能.

3 结束语

本文通过改进加权最小连接调度算法, 周期性地采集存储节点的 CPU、内存、硬盘、I/O 利用率信息, 并结合节点的权值, 计算出每个节点的当前负载情况, 准确地选择负载最轻的节点并分发任务, 有效地避免了节点超载后, 负载均衡器依然向节点发送服务连接请求, 从而引发存储读写速度下降的问题, 也提升了存储读写性能、数据吞吐率.

但是, 由于节点的权值会随着节点接收的任务数的增加而出现较大的偏差. 当节点的 CPU、内存、硬盘、I/O 利用率与任务连接数的比例相当时, 此时的负载均衡调整会重点根据权值选择节点分发任务. 如果权值有偏差可能会导致负载不均衡、调度算法循环调度等问题产生, 造成资源浪费、时间开销过大.

因此, 在改进的加权最小连接数调度算法中如何根据 CPU、内存、硬盘、I/O 利用率与任务连接数的统计值更合理动态调整权值, 以解决权值动态调整受限问题, 需在今后的工作和学习中进一步研究与探索.

参考文献

1 Zhao Y, Huang WL. Adaptive distributed load balancing algorithm based on live migration of virtual machines in cloud. Fifth International Joint Conference on INC, IMS and IDC, 2009. Seoul, South Korea. 2009. 170–175.

2 胡丽聪, 徐雅静, 徐惠民. 基于动态反馈的一致性哈希负载均衡算法. 微电子学与计算机, 2012, 29(1): 177–180.

3 张玉芳, 魏钦磊, 赵膺. 基于负载权值的负载均衡算法. 计算机应用研究, 2012, 29(12): 4711–4713. [doi: 10.3969/j.issn.1001-3695.2012.12.080]

4 刘健, 徐磊, 张维明. 基于动态反馈的负载均衡算法. 计算机工程与科学, 2003, 25(5): 65–68.

5 Pan JS, Wang HB, Zhao HN, *et al*. Interaction artificial bee colony based load balance method in cloud computing. Sun H, Yang CY, Lin CW, *et al*. Genetic and Evolutionary Computing. Cham: Springer International Publishing, 2015. 49–57.

6 杨石, 王艳玲, 王永利. 云计算环境下基于蜜蜂觅食行为的任务负载均衡算法. 计算机应用, 2015, 35(4): 938–943. [doi: 10.11772/j.issn.1001-9081.2015.04.0938]

7 Nakai AM, Madeira E, Buzato LE. Load balancing for internet distributed services using limited redirection rates. 2011 5th Latin-American Symposium on Dependable Computing (LADC). Sao Jose dos Campos, Brazil. 2011. 156–165.

8 郭平, 李琪. 基于服务器负载状况分类的负载均衡调度算法. 华中科技大学学报 (自然科学版), 2012, 40(S1): 62–65.

9 陈亮, 王加阳. 基于粗糙集的负载均衡算法研究. 计算机工程与科学, 2010, 32(1): 101–104.

10 Rittinghouse JW. Ransome JF. 云计算. 田思源, 赵学锋译. 北京: 机械工业出版社, 2010.

11 赵丹丹. Swift 的读取负载均衡研究与实现[硕士学位论文]. 南京: 东南大学, 2016.

12 Azar Y, Broder AZ, Karlin AR. On-line load balancing. Theoretical Computer Science, 1994, 130(1): 73–84. [doi: 10.1016/0304-3975(94)90153-8]

13 Lee R, Jeng B. Load-balancing tactics in cloud. 2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (Cyber C). Beijing, China. 2011. 447–454.

14 Bunt RB, Eager DL, Oster GM. *et al*. Achieving load balance and effective caching in clustered web servers. Proceedings of the 4th International Web Caching Workshop. San Diego, CA, USA. 1999.

15 杨锦, 李肯立, 吴帆. 异构分布式系统的负载均衡调度算法. 计算机工程, 2012, 38(2): 166–168.

16 熊振华. 基于 OpenStack 云存储技术的研究[硕士学位论文]. 长春: 吉林大学, 2014.

17 Chang H, Tang XH. A load-balance based resource-scheduling algorithm under cloud computing environment. Proceedings of the 2010 International Conference on New Horizons in Web-Based Learning. Shanghai, China. 2011. 85–90.

© 中国科学院软件研究所 http://www.c-s-a.org.cn