2009 年 4 月 第 4 期
Vol. 30 No. 4 2009

# 一种元数据服务器集群的负载均衡算法

王 娟, 冯 丹, 王 芳, 廖振松

(华中科技大学 信息存储系统教育部重点实验室, 湖北 武汉 430074)
E-mail: rabbitman@126.com

**摘 要**: 在基于对象的存储系统中, 元数据访问非常频繁, 大规模存储系统中元数据的访问是潜在的系统性能瓶颈. 元数据服务器集群中必须负载均衡, 以防某个元数据服务器成为存储系统访问的瓶颈. 现有文章中很少有研究元数据服务器集群的负载均衡的文章. 本文中采用元数据请求的响应时间来衡量一个元数据服务器的负载情况, 首先从映射算法上实现静态负载均衡, 并针对元数据热度差别大而引起的负载不均衡引入动态负载均衡, 通过仿真结果显示其有效性.

**关 键 词**: 基于对象的存储; 元数据服务器集群; 负载均衡; 请求响应时间
**中图分类号**: TP393 **文献标识码**: A **文 章 编 号**: 1000-1220(2009)04-0757-04

## Load Balancing Algorithm in MetaData Server Cluster

WANG Juan, FENG Dan, WANG Fang, LIAO Zhen-song
(*Key Laboratory of Data Storage System, Ministry of Education, Huazhong University of Science and Technology, Wuhan* 430074, *China*)

**Abstract**: In object-based storage system, metadata is frequently accessed, which is the underlying storage system performance bottleneck. Load balancing must be implemented in metadata server cluster in order to avoid any metadata server becomes the bottleneck of the system. Few existing papers are about load balancing in metadata server cluster. This paper uses metadata request response time for measuring the load condition of a metadata server. A mapping algorithm is adopted at first to achieve the static load balancing, and the dynamic load balancing algorithm is introduced to solve the load imbalance brought by the difference of metadata popularity.

**Key words**: object-based storage; metadata server cluster; load balancing; request response time

## 1 引 言

随着计算机技术的发展和应用的普及, 信息存储容量成爆炸性地增长, 现有网络存储系统已无法满足人们对于存储的需要. 对象存储(Object-Based Storage, OBS)技术应运而生, 它利用现有的处理技术、网络技术和存储组件, 可以通过一种简单便利的方式来获得前所未有的可扩展性和高吞吐量[1], 成为下一代网络存储的主流.

为了获得对象存储系统的高性能和高可伸缩性, 避免瓶颈是非常重要的. 虽然文件元数据的大小相对于对象数据来说只占很小的空间, 然而由于文件元数据访问非常频繁, 因此元数据的访问是个潜在的瓶颈, 管理好元数据是获取存储系统高性能和高伸缩性的前提. 随着对象存储系统规模的增大以及文件元数据访问流行度的增加, 系统中需要有多个元数据服务器(MetaData Server, MDS), 它们构成MDS 集群. MDS 集群要提供高性能、可扩展的元数据服务, 因此它们之间必须负载均衡, 以防任意一个MDS 成为系统访问的瓶颈.

当前的负载的定义有多种标准, 对于存放数据的设备来说, 由于不同的应用的性能可能会受限于不同的组件, 单独从一个方面来衡量设备的负载是不合适的, 因此对一个设备负载的评估要考虑多种因素: 如请求队列长度、CPU、I/O 处理能力、网卡速度等, 任何一个组件阻塞都有可能成为系统的瓶颈, 因此文章[2, 3]将这些参数通过计算系数加权成负载权值来衡量一个设备的负载, 然而这种方法的计算系数的选择非常困难, 如果系数选择不适当, 反而会使系统的性能更差. 当前大部分文章都是对存放数据的设备的负载均衡进行的研究[6, 7, 8], 存放数据的设备的负载情况与MDS 集群的负载情况有很大的差别, 很少有文章研究MDS 集群的负载情况. 对于MDS 集群来说, 它仅仅为用户提供文件元数据请求服务, 文件元数据的大小很小, 请求率很高, 本文采用文件元数据请求的响应时间来衡量一个MDS 的负载情况. 在本文中, 首先通过对文件名的哈希, 将文件元数据均匀的分配在不同的MDS 上, 从而实现MDS 集群中的MDS 之间的静态负载均衡. 然而由于应用负载随时变化, 每个文件元数据被访问的频率(热度)差别很大, 因此可能会导致某个MDS 上的负载过重, 成为系统瓶颈, 导致整体系统性能下降. 针对这种情况, 作者引入动态负载均衡, 将负载重的MDS 上的部分文件元数据转移到负载轻的MDS 上.

收稿日期: 2007-11-20 基金项目: "九七三"国家重点基础研究发展规划项目(2004CB318201)资助; 国家自然科学基金项目(60503059)资助; "新世纪优秀人才支持计划"项目(NCET-04-0693 和NCET-06-0650)资助. 作者简介: 王 娟, 女, 1981 年生, 博士研究生, 研究方向为网络存储技术; 冯 丹, 女, 1970 年生, 教授, 博士生导师, 研究方向为信息存储理论和系统结构; 王 芳, 女, 1972 年生, 副教授, 研究方向为信息存储理论和系统结构; 廖振松, 男, 1979 年生, 博士研究生, 研究方向为网络安全、信息安全.

## 2 负载均衡策略

### 2.1 静态负载均衡

在MDS 集群中, 首先通过预先制定的文件元数据到MDS 的映射算法, 将系统中的文件元数据均匀的分布到不同的MDS 中.

对象存储系统中的MDS 集群的集合为M (n),MDS 的个数为 ‖M (n) ‖ = N;

文件元数据到MDS 的映射算法如下:

(1) val= Hash(POID, filename);

(2)Local(val) = m ; 并且m ∈M (n);

其中, POID 代表该文件所属父目录的对象标识符, filename 代表该文件元数据的文件名, Hash 代表一个哈希函数, 将该文件名映射成一个值val;Local 代表一个映射函数, 将文件名哈希值为val 的文件元数据存放到MDS 集群M (n) 中序号为m 的MDS 中. 由于选择的哈希函数Hash 是伪随机均匀分布的, 映射函数Local 是均匀分布的, 所以最终系统中的文件元数据在MDS 集群中是均匀分布的. 即如果每个文件的访问频率相同的话, 则每个MDS 上的访问负载是近似相同的.

### 2.2 动态负载均衡

由于对象存储系统为不同的应用服务, 应用负载随时都可能发生较大的改变致使每个文件元数据被访问的频率(热度)差别很大, 使得MDS 节点之间负载不均衡. 本文通过判断MDS 集群中每个MDS 的请求响应时间与整个MDS 集群的平均响应时间的差异来判断MDS 负载是否均衡. 当MDS 集群中MDS 间负载不均衡时, 应该采用动态负载均衡, 将负载重的MDS 上的部分文件元数据复制或转移到负载轻的MDS 上.

#### 2.2.1 MDS 的请求的响应时间

假定序号为x 的MDS 对文件元数据请求的响应时间为R (x) (x∈M (n));

一般认为, 每个MDS 可以采用M/G/1 排队模型来建模[4]. 根据排队论可得MDSx 对文件元数据请求的响应时间为:

$$R(x)=\frac{\lambda_x * E(T_x^2)}{2*(1-\rho_x)}+E(T_x) \tag{1}$$

其中:

$$\lambda_x=\sum_{k\in A(x)}\lambda_{xk};\quad \rho_x=\sum_{k\in A(x)}\lambda_{xk}*T_{xk};$$

$$E(T_x)=\sum_{k\in A(x)}p_{xk}*T_{xk}=\sum_{k\in A(x)}\frac{\lambda_{xk}}{\lambda_x}*\frac{L_{xk}}{B_x};$$

$$E(T_x^2)=\sum_{k\in A(x)}p_{xk}*T_{xk}^2=\sum_{k\in A(x)}\frac{\lambda_{xk}}{\lambda_x}*\left(\frac{L_{xk}}{B_x}\right)^2.$$

公式中的符号定义如下所示

- $T_x$:$MDS_x$ 上的元数据请求服务时间;
- $\lambda_x$:$MDS_x$ 上的元数据请求到达率;
- $\rho_x$:$MDS_x$ 上的访问强度;
- $B_x$:$MDS_x$ 的处理能力;
- A (x):$MDS_x$ 上收到的文件元数据请求的集合;
- $\lambda_{xk}$:$MDS_x$ 上的文件k 的元数据请求到达率;
- $T_{xk}$:$MDS_x$ 上的文件k 的元数据请求服务时间;
- $p_{xk}$:$MDS_x$ 上的文件k 被访问的概率;
- $L_{xk}$:$MDS_x$ 上的文件k 的元数据长度;

由于MDS 中每个文件的元数据都是结构化的数据, 大小都相等. 同时假设每个MDS 的处理能力都相同, 则每个MDS 对每个文件元数据的请求服务时间都相同, 即对∀x∈M (n), k∈A (x) 有 $T_{xk}=\frac{L_{xk}}{B_x}=C$ (C 为常数); 则式(1) 最终可化简为:

$$R(x)=\frac{\lambda_x * C^2}{2*(1-\lambda_x C)}+C \tag{2}$$

其中, 该MDS 集群的平均访问强度为:

$$\rho(R)=\frac{1}{N}\sum_{i=1}^{N}\rho_i \tag{3}$$

该MDS 集群的平均响应时间为:

$$E(R)=\frac{1}{N}\sum_{i=1}^{N}R(i) \tag{4}$$

该MDS 集群中的响应时间方差为:

$$D(R)=\frac{1}{N}\sum_{i=1}^{N}(E(R)-R(i))^2 \tag{5}$$

#### 2.2.2 MDS 动态负载均衡的算法

在对象存储系统中,MDS 之间的负载均衡属于完全分布式的MDS节点之间需要周期性的交换它们相互的负载信息.

MDS 动态负载均衡的目标是使每个MDS 对文件元数据请求的响应时间相差不大, 本文用下面的函数来衡量$MDS_x$ 的负载不均衡度:

$$DM(x)=R(x)-E(R)\quad (x\in M(n)) \tag{6}$$

函数DM (x) 是指$MDS_x$ 请求响应时间与MDS 集群的平均响应时间的差异, 它的大小反映了$MDS_x$ 负载的均衡性.

首先定义一个负载差阈值$DM_{ALARM}$, 在MDS 集群中, 如果∃∀x∈M (n) 使得DM (x) ≥$DM_{ALARM}$ 时, 则将在MDS 上启动下列动态负载均衡算法.

具体算法描述如下:

```
L_MDS = Φ; /* 负载较轻的MDS 的集合 */
W_MDS = Φ; /* 负载过载的MDS 的集合 */
1) for i = 1 TO N
     if(DM(i) ≥DM_ALARM)
        W_MDS = W_WDS∪{i};
     elseif(DM(i) < 0)
        L_MDS = L_WDS∪{i};
2) while(W_MDS≠Φ)
   MDS_i = max({DM(x) | x∈W_MDS}), 找一个子集
   M_MDS ⊆ L_MDS, 满足如下条件:
```

$$\left|DM(i)+\sum_{j\in M\_MDS}DM(j)\right|\le\left|DM(i)+\sum_{k\in SL\_MDS}DM(k)\right|,$$

$\forall SL\_MDS \subseteq L\_MDS$

3) $\forall MDS_j \in M\_MDS$, 将$MDS_i$ 中的部分文件元数据传输到$MDS_j$ 上; $MDS_i$ 和$MDS_j$ 之间传输的文件元数据集合的总的请求到达率大约为:

$$\frac{2 * (E(R) - C)}{(2 * E(R) - C) * C} - \lambda_j$$

4) W_MDS = W_WDS-{i};
L_MDS = L_WDS - M_WDS;
if (W_MDS ≠Φ) goto 2).

在上述算法中, 并不是对超过平均响应时间的每个MDS 都启动动态负载均衡, 而是当其值超过一定的阈值$DM_{ALARM}$ 时 才启动, 这样可以减少文件元数据在MDS 之间频繁的迁移, 而只是在必要的时候才迁移 启动动态负载均衡后, 从负载过载最多的$MDS_i$ 开始, , 将$MDS_i$ 中过载部分的文件元数据按照一定的比例分配到合适的负载较轻的MDS 中, 最终使得系统中的MDS 负载均衡

## 3　算法仿真结果

作者编写仿真程序验证上述负载均衡算法 其中假设MDS 集群中MDS 的个数N = 4, 系统中文件元数据请求服从泊松到达 为了模拟文件元数据请求的重负载, 仿真程序在MDS 集群中不断创建新的文件元数据, 文件元数据一旦创建后就留在MDS 中并被应用客户端访问 随着文件元数据请求的增多, $MDS_x$ (x= 1, 2, 3, 4) 上的访问强度$\rho_x$ 也不断增加, 仿真程序一直运行直至MDS 集群的平均访问强度$\rho(R)$ 至0. 9 (因为每个MDS 负载可能不均衡, 当系统平均访问强度为该值时, 实际上重负载的MDS 已经超过了该值并几乎接近1).

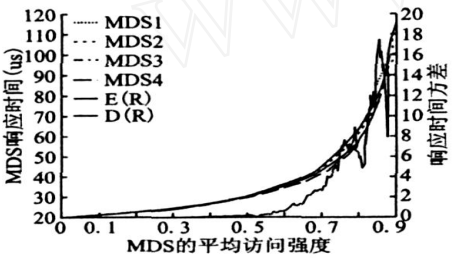



图1 元数据请求到达率相同情况下, 静态负载均衡时的MDS 的响应时间

Fig. 1　Etadata request response time of each MDS when uses static load balancing algorithm with the same metadata request arriving rate

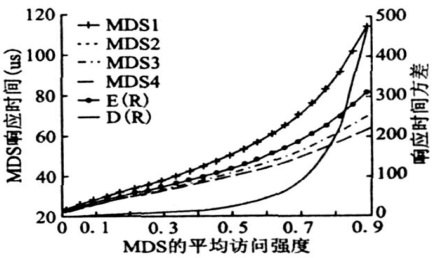



图2 元数据请求服从zipf 分布时的, MDS 的响应时间

Fig. 2　Metadata request response time of each MDS with the metadata request arriving rate obeying the zipf distribution

设MDS 集群中被访问的文件元数据个数为$\sum_{x=1}^{N} A(x) = X$, 被访问的文件元数据f(i) 的请求到达率为$\lambda_{f(i)}$, 为了保证MDS 集群的平均访问强度不大于0. 9, 则$\rho(R) = \frac{1}{N}\sum_{i=1}^{N}\rho_i = \frac{1}{N}\sum_{i=1}^{N}(C * \sum_{k\in A(x)}\lambda_k) \le 0.9$ 即$\sum_{i=1}^{X}\lambda_{f(i)} \le 0.9 * \frac{N}{C}$. 假设每个MDS 的中每个文件元数据的服务时间为: C= 0. 02ms, 则有4 个MDS 组成的MDS 集群中要保证被访问的总的文件元数据到达率$\sum_{i=1}^{X}\lambda_{f(i)} \le 0.9 * (2 * 10^5)$ (请求/秒).

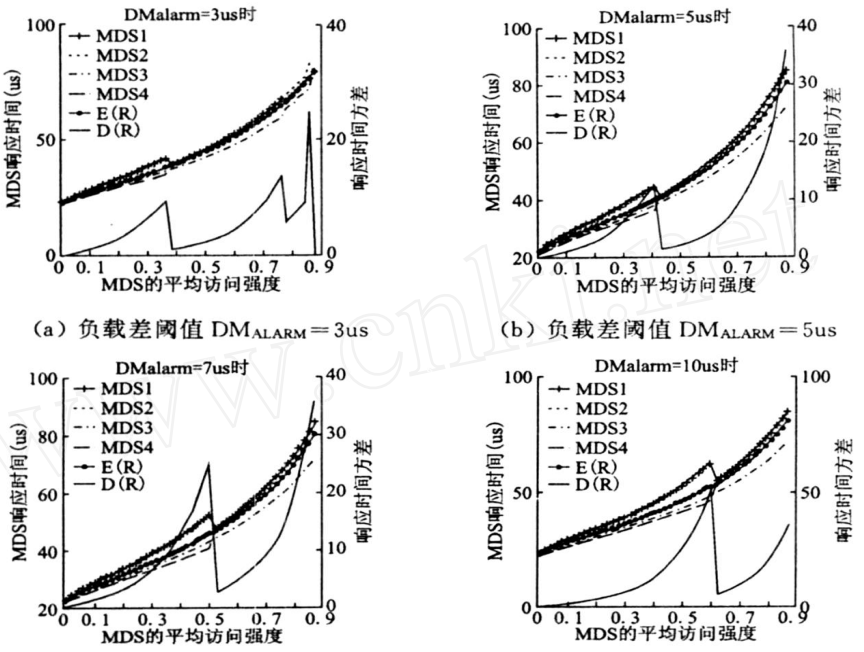
首先来看一下, 在采用静态负载均衡后, MDS 集群中MDS 响应时间情况 从图2 中可以看出, 在每个文件元数据请求到达率相同的情况下, 采用静态负载均衡的系统的每个MDS的平均响应时间差别微小, MDS响应时间方差的值很小.

为了更好地说明问题, 在下面的仿真中, 假定按照静态负载均衡, 文件元数据是完全均匀分布在MDS 集群中的 下面给出在采用静态负载均衡时, 当应用负载发生变化, 即文件热度不同时, MDS 集群中MDS 的响应时间情况 假定文件元数据请求到达率服从Zipf 分布[5] (一般认为, 媒体文件的访问概率服从该分布), 则在该负载下的结果如图3 所示

由此可见, 在采用了静态负载均衡后, 虽然文件元数据被均匀分布在不同的MDS 上, 然而在文件元数据访问热度相差较大的情况下, 每个MDS 上的负载差别比较大, 导致MDS 之间的响应时间差别很大 这个时候需要引入动态负载均衡, 将负载重的MDS上的部分文件元数据转移到负载轻的MDS上

在仿真程序中分别设置负载差阈值$DM_{ALARM}$ 为3us、5us、7us, 10us 时, 采用动态负载均衡时的MDS 集群的响应时间分布如图3 所示

由图3 可以看出负载差阈值$DM_{ALARM}$ 的设置决定了MDS 之间的均衡度, 当$DM_{ALARM}$ 较小时, MDS 集群中的响应时间方差较小, MDS 之间负载比较均衡, MDS 之间的迁移次数就比较多. 当$DM_{ALARM}$ 较大时, MDS 集群中的响应时间方差较大, MDS 之间的迁移次数较少. 因此要合适的选择一个$DM_{ALARM}$ 的值, 作者最终选取了$DM_{ALARM}$ = 10us 综上所述, 对比图2 可以看出, 在文件元数据访问频率差别很大的情况下, 启动动态负载均衡算法能很大程度地减少MDS 集群中的MDS 的响应时间的差别, 大大减少系统的响应时间方差




(a) 负载差阈值 $DM_{ALARM}=3us$ (b) 负载差阈值 $DM_{ALARM}=5us$

(c) 负载差阈值 $DM_{ALARM}=7us$ (d) 负载差阈值 $DM_{ALARM}=10us$

图 3 在与图 2 相同的请求到达率分布情况下,采用动态负载均衡后的结果

Fig. 3 Metadata request response time of each MDS when uses the dynamic load balancing algorithm with the metadata request arriving rate obeying the zipf distribution

## 4 结束语

本文提出一种应用于MDS集群的负载均衡算法 首先从映射算法上实现MDS集群的静态负载均衡 而针对由于文件元数据热度差别大而引起的负载不均衡,则引入动态负载均衡,从而使得整个MDS集群中的MDS的响应时间差别较小,达到MDS集群的负载均衡 当前本文中只考虑了仿真的情况,并且为了试验方便,假定文件元数据请求到达率服从一定的分布,实际上负载是很多应用负载的合成,请求到达率的分布并没有这么规则,下一步打算采用实际的 trace 对该算法进行测试

**References:**


[1] Mike Mesnier, Gregory R Ganger, Erik Riedel. Object-based storage[J]. IEEE Communications Magazine, 2003, 41(8): 84-90.

[2] Shan Zhi-guang, Dai Qiong-hai, Lin Chuang, et al. Integrated schemes of Web request dispatching and selecting and their performance analysis[J]. Journal of Software, 2001, 12(3): 355-366.

[3] Yu Lei, Lin Zong-kai, Guo Yu-chai, et al. Load balancing and fault-tolerant services in multi-server system[J]. Journal of System Simulation, 2001, 13(3): 325-328.

[4] Sun Rong-heng, Li Jian-ping. On the basis of the queue[M]. Beijing: Science Press, 2002.

[5] Michael Rabinovich, Irina Rabinovich, Rajmohan Rajaraman, et al. A dynamic object replication and migration protocol for an Internet hosting service[C]. 19th IEEE International Conference on Distributed Computing Systems, Austin, Texas, USA, 1999.

[6] Qin Ling-jun, Feng Dan, Zeng Ling-fang, et al. Dynamic load balancing algorithm in object-based storage system[J]. Computer Science, 2006, 33(5): 88-91.

[7] 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.

[8] Ni Yun-zhu, Lu Guang-hong, Huang Yan-hui. The solution of disk load balancing based on disk striping with genetic algorithm[J]. Chinese Journal of Computers, 2006, 29(11): 1995-2002.


**附中文参考文献:**


[2] 单志广, 戴琼海, 林 闯, 等. Web 请求分配和选择的综合方案与性能分析[J]. 软件学报, 2001, 12(3): 355-366.

[3] 于 磊, 林宗楷, 郭玉钗, 等. 多服务器系统中的负载平衡与容错[J]. 系统仿真学报, 2001, 13(3): 325-328.

[4] 孙荣恒, 李建平. 排队论基础[M]. 北京: 科学出版社, 2002.

[6] 覃灵军, 冯 丹, 曾令仿, 等. 基于对象存储系统的动态负载均衡算法[J]. 计算机科学, 2006, 33(5): 88-91.

[7] 郭成城, 晏蒲柳. 一种异构Web服务器集群动态负载均衡算法[J]. 计算机学报, 2005, 28(2): 179-184.

[8] 倪云竹, 吕光宏, 黄彦辉. 用遗传算法解决基于分条技术的磁盘负载均衡问题[J]. 计算机学报, 2006, 29(11): 1995-2002.