软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(2):331−342 [doi: 10.3724/SP.J.1001.2013.04177]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

对象存储系统中自适应的元数据负载均衡机制*

陈 涛[1,2,3,4], 肖 侬[1], 刘 芳[1]

[1](国防科学技术大学 计算机学院,湖南 长沙 410073)

[2](事医学科学院 放射与辐射医学研究所,北京 100850)

[3](蛋白质组学国家重点实验室(军事医学科学院),北京 102206)

[4](北京蛋白质组研究中心,北京 102206)

通讯作者: 陈涛, E-mail: lovely696521@163.com

摘 要: 面向对象的存储系统在研究、工程以及服务领域均得到了广泛的应用.在面向对象的存储系统中,元数据的负载均衡对于提高整个系统的 I/O 性能具有重要的作用.现有的元数据负载均衡策略不能动态地平衡元数据的访问负载,而且自适应性以及容错特性有待提高.提出了一种自适应的分布式元数据负载均衡机制(adaptable distributed load balancing of metadata,简称 ADMLB),包含基本的负载均衡算法和分布式的增量负载均衡算法.采用基本的负载均衡算法按照服务器的性能公平地分布负载,使用分布式的负载均衡算法定时地调整负载的分布.ADMLB 采取分布式的方法均衡地在元数据服务器之间分布负载,根据负载的变化自适应地进行调整,具有很好的容错特性,而且用户可以高效地定位元数据服务器.

关键词: 面向对象的存储系统;元数据;负载均衡;自适应;分布式

中图法分类号: TP316 **文献标识码**: A

中文引用格式: 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.软件学报,2013,24(2):331−342. http://www.jos.org.cn/1000-9825/4177.htm

英文引用格式: Chen T, Xiao N, Liu F. Adaptive metadata load balancing for object storage systems. Ruanjian Xuebao/Journal of Software, 2013,24(2):331−342 (in Chinese). http://www.jos.org.cn/1000-9825/4177.htm

Adaptive Metadata Load Balancing for Object Storage Systems

CHEN Tao[1,2,3,4], XIAO Nong[1], LIU Fang[1]

[1](College of Computer, National University of Defense Technology, Changsha 410073, China)

[2](Beijing Institute of Radiation Medicine, Beijing 100850, China)

[3](State Key Laboratory of Proteomics, Beijing 102206, China)

[4](Beijing Proteome Research Center, Beijing 102206, China)

Corresponding author: CHEN Tao, E-mail: lovely696521@163.com

Abstract: Object-Based storage is a good choice for large scale storage systems. Load balancing of metadata is important to improve the performance of I/O. The existing load balancing schemas cannot evenly distribute the accesses of metadata in a dynamic way. Moreover, the adaptability and fault-tolerance ability need to be improved. This paper presents an adaptable distributed load balancing of metadata (ADMLB) which is composed of basic load balancing algorithm (BBLA) and distributed incremental load balancing algorithm (IBLA). Specially, ADMLB first uses BBLA to distribute metadata loads according to the performances of the metadata servers and then uses IBLA to incrementally reorganize loads on each metadata server. ADMLB can evenly distribute loads between metadata servers and adapts well to the changes of loads. It also has good fault-tolerance ability, and locates metadata servers very quickly.

Key words: object storage system; metadata; load balancing; adaptive; distributed

* 基金项目: 国家自然科学基金(61025009, 60903040, 61070198, 61170288); 国家高技术研究发展计划(863)(2011AA010500)
收稿时间: 2010-09-20; 修改时间: 2011-03-29; 定稿时间: 2011-09-01

无论在生物信息、核能、航空航天等科学研究与工程领域,还是在教育、媒体、金融、医疗设备等信息服务领域,日益增长的海量数据的存储与管理都面临着巨大的挑战.传统的体系结构,如直接附加存储 DAS 和网络附加存储 NAS,由于其容量以及 I/O 性能的限制,已经无法满足海量数据存储的需求.存储区域网络 SAN 使用专用存储网络连接主机和设备,可以提供几乎不受设备数量限制的存储容量,适合存储大量数据.但是底层磁盘设备无法验证用户的身份,安全性很差.面向对象的存储系统不仅可以提供海量信息的存储,而且底层的智能存储设备(object storage device,简称 OSD)拥有 CPU、内存和网络接口,具有身份认证的功能,可以满足海量数据存储应用的需求.

在面向对象的存储系统中,文件的元数据和数据分开存储,元数据的存储和管理由元数据服务器(metadata server,简称 MDS)负责,而数据存储在 OSD 上.用户从 MDS 获取文件的元数据后,直接与 OSD 通信,数据的传输不需要经过 MDS 转发.尽管元数据的数据量相对于整个存储系统的数据容量而言比较小,但是在访问数据之前必须先访问元数据.统计数据表明,元数据的访问操作占整个系统访问操作的 50%~80%,因而元数据管理的性能直接影响到整个 I/O 性能.集中式的元数据管理容易形成性能瓶颈,而且由于元数据的负载不断变化,分布式以及自适应的元数据管理成为发展的趋势.有效的元数据负载均衡机制是实现分布式以及自适应元数据管理的关键技术之一.分布式的元数据管理使用一个共享存储的服务器集群来响应用户的元数据操作,共享的 OSD 设备负责具体的元数据存储,集群里的每个服务器可以访问所有的 OSD 设备.这种共享存储的元数据集群具有很好的失效恢复机制.由于服务器可以访问所有的元数据,当一个服务器失效时,其他服务器可以承担该服务器的负载.元数据的负载均衡策略研究的问题是如何将元数据负载公平地分布到元数据服务器集群上,使得元数据服务器集群的整体性能最大化,同时自适应元数据负载的变化.

目前,元数据的负载均衡策略不能有效地平衡元数据的访问负载,而且在自适应性以及容错特性上存在严重不足.针对现有方法的缺陷,本文针对面向对象存储系统中共享 OSD 设备的元数据服务器集群,提出了一种自适应的分布式元数据负载均衡机制(adaptable distributed load balancing of metadata,简称 ADMLB),包括基本的负载均衡算法和分布式的增量负载均衡算法.基本的负载均衡算法采用基于权重的 hash 方法,在系统初建时将元数据负载大小视为相同,然后按照该方法,在异构的服务器之间根据服务器性能均衡地分布元数据负载.分布式的增量负载均衡算法采用 FAST TCP 中经典的拥塞控制函数定时地更新服务器的性能,此时,服务器的性能不仅体现了服务器本身的异构性,而且考虑了服务器上元数据的负载大小,即综合考虑元数据负载以及服务器的异构性.根据更新的服务器性能再次使用基本的负载均衡算法重新调整负载的分布情况,使得负载能够重新公平地分布在服务器上.本文提出的元数据负载均衡机制具有以下优点:

(1) 可以处理异构的元数据负载.ADMLB 机制根据负载的大小调整其在服务器之间的分布.

(2) 根据服务器的性能异构特性公平地分布负载.服务器的性能存在差异,ADMLB 机制考虑了服务器的异构性,按照服务器的性能来均衡负载.

(3) 自适应元数据负载的增加和删除.当用户写数据时,首先要创建元数据,根据系统当时的元数据服务器性能配置情况,使用基本的元数据负载均衡算法分配新增的元数据.系统会定时地使用增量式的元数据负载均衡算法来调整负载分布,避免增加元数据负载导致的短暂的负载不公平性.我们注意到,在大量增加元数据时,根据基本的负载均衡算法的 hash 特性,新增元数据负载仍然能够均衡地分布在服务器之间.在删除元数据时,根据基本的负载均衡算法计算负责该元数据的服务器,由该元数据服务器清除 cache 里的缓冲数据,并删除相应 OSD 上的元数据信息.同时,在下一个周期内调整元数据负载的分布.

(4) 自适应元数据服务器规模的变化.当增加新的服务器时,调整各个服务器的权值,只需从旧服务器上迁移相应的负载到新增服务器上,服务器之间没有具体的数据迁移.相应地,在删除旧的服务器时,根据服务器的新权值,将该服务器上的负载转移到其他服务器上.

(5) 具有很好的容错特性.采用共享存储的元数据服务器集群的体系结构,所有的服务器均可以访问底层的 OSD 设备.当一个服务器失效时,该服务器上的负载可由其他服务器来承担.ADMLB 机制使得

失效服务器的负载按照其他服务器的性能均衡地分布,具有很好的容错特性.

(6) 可以快速地计算某个元数据负载所在的服务器,使用一个预定义的函数在 $O(\log n)$内计算某个元数据负载所在的服务器,其中,n 表示服务器的个数.

本文第 1 节描述相关工作.第 2 节给出模型以及相关概念.第 3 节提出自适应的分布式负载均衡机制 ADMLB,详细描述该机制的两个组成部分:基本的负载均衡算法以及分布式的增量负载均衡算法.第 4 节对 ADMLB 机制进行理论分析,阐述 ADMLB 机制的各种特性.第 5 节给出实验与结果分析.第 6 节总结全文.

1 相关工作

使用单一的元数据服务器来提供元数据服务时,可扩展性受到很大的限制,容易形成性能瓶颈,而且存在单点失效.Lustre[1]使用了两个元数据服务器,其中一个服务器是另一个服务器的完全备份,可以避免单点失效的问题,但是可扩展性以及性能有待提高.单一的元数据服务器或者如 Lustre 使用的两个备份的元数据服务器都不存在负载均衡的问题.为了适应科学计算以及其他数据密集型应用的需求,需要多个元数据服务器来管理海量数据的元数据.本文探讨的元数据负载均衡机制研究如何在元数据服务器集群之间公平地分布元数据负载.现有的研究主要集中于子树分割与 hashing 这两类方法.

子树分割分为静态和动态两种:静态子树分割是一种自然的负载均衡方法,按照文件的目录树来组织文件的元数据,将目录树的一个或者多个子目录树分配给一个元数据服务器.用户根据文件的目录路径确定其元数据所在的目录子树,确定与该子树相应的元数据服务器.子树一旦分割就不会改变,不能适应元数据负载的变化以及服务器规模的改变,如 NFS[2],Andrew[3],Coda[4]等都是采用这种方法来管理元数据;动态子树分割[5]是一种基于静态子树划分的改进方法,也是将目录树的不同子树委托授权到不同的元数据服务器,并且允许同一子树下的不同子树嵌套委托,根据元数据服务器的负载变化将不同的子树动态地委托给元数据服务器,支持元数据负载变化.该方法仍然无法均衡元数据负载.无论是静态还是动态子树分割,都不能有效地平衡元数据负载.文献[6]提出了一种动态规划的方法以及二叉树搜索方法来解决元数据的分割问题,将元数据的某个子集分到元数据服务器集群上,但是该方法没有考虑元数据负载的变化.动态 Dir-Grain 方法将名字空间动态地划分为大小可调节的层次单元,从而保持名字空间的局部性,均匀分布负载.但是,动态划分名字空间的方法也没有服务器的增加和删除情况[7].

Hashing 方法主要有静态 hashing、延迟混合[8]、目录路径 hashing[9]和动态 hashing[10].静态 hashing 根据 hash 值在服务器之间分配元数据,不能根据服务器的性能均衡地分配负载,不支持服务器规模的改变.延迟混合[8]在静态 hashing 的基础上进行改进,基于文件全路径名进行 hash 值的计算.同时,延迟元数据更新的方法是,将元数据的更新、迁移延迟到需要访问的时候,将元数据更新与迁移开销分摊到不同的时间.延迟混合方法不支持服务器的异构性,在服务器规模变化时,几乎所有 hash 值都会发生变化.目录路径方法[9]将全文件路径名分为目录路径和文件名两部分,使用目录路径服务器来管理目录元数据,使用元数据服务器来管理文件元数据.单一的目录路径服务器是一个性能瓶颈,而且对文件名的 hash 计算无法保证动态的负载均衡.动态 hashing 方法[10]使用文件全路径名 Hash 来分配元数据到服务器集群上,采用相对负载平衡策略自适应地均衡元数据访问负载,弹性策略支持服务器的增删及替换.以上所有的 hashing 方法都无法根据元数据服务器的性能均衡地分配元数据负载.

Zhu 和 Hua 等人使用 Bloom Filter 技术高效地定位元数据服务器[11,12],没有考虑元数据的分配以及负载均衡问题.

目前,相关研究关注于存放数据的服务器或者磁盘之间均衡数据负载[13,14].服务器或者磁盘的数据负载与本文研究的元数据服务器集群上元数据负载存在较大差异.因为数据是大 I/O,元数据是小 I/O,它们在访问特征以及访问模式上存在很大的不同.在前期的工作中,我们提出了基于聚类和一致 hash 的数据布局算法,可以在各个设备间公平地分布数据[15].该方法充分考虑了设备之间的异构特性,将设备分为多个集群,在集群之间以及集群内部进行数据的二次分配.其他研究从多方面的因素来综合考虑负载,如请求队列长度、CPU、I/O 处理能力

等[16],若参数选择不当,性能反而下降.本文采取元数据服务器的访问延迟来体现元数据负载的变化,避免参数的多选择问题.

DHT 是一种分布式存储方法,每个节点负责一个小范围的路由,并负责存储一小部分数据,从而实现整个 DHT 网络的寻址和存储.DHT 是一种概念,例如,一致性 hash 是 DHT 的一种实现方式,但是一致性 hash 只能适应同构的情况,将数据看作是等同的.而在元数据负载的分布问题中,负载的大小是异构的,一致性 hash 无法解决异构负载以及异构服务器上的负载分布问题.本文通过调整服务器的权值,体现负载的异构特征,最终使得每个服务器的访问延迟与所有服务器的平均访问延迟相同,使得异构负载在异构服务器间公平分布.在某些 DHT 的实现方法中,节点以表的形式保存其他节点的路由信息.在定位数据所在的位置时,需要查询其他节点的路由表.本文的方法在定位负载时,只需获取特定的参数,使用函数可以直接计算负载所在的服务器.

本文提出的元数据负载均衡机制由两部分组成:基本的负载均衡算法和分布式的增量负载均衡算法.基本的负载均衡算法是采用基于权重的分布式 hashing,根据服务器的性能公平地分配负载.分布式的增量负载均衡算法使用 FAST TCP 里的控制函数按照服务器的访问延迟动态地调整负载.FAST TCP 是近期出现的 TCP 的改进方法,使用包的延迟代替传统的包丢失率来进行通塞控制[17].本文提出的元数据负载均衡机制可以自适应负载的变化,支持服务器的增加和删除,同时具有很好的容错特性,失效服务器的负载能够均衡地分布到其他服务器上.使用预定义的函数可以在 $O(\log n)$时间内快速地定位元数据负载所在的服务器.

2 模型及相关概念

共享 OSD 设备的元数据服务器集群的体系结构如图 1 所示.共享 OSD 设备负责存储元数据,MDS 处理来自用户的元数据请求.MDS 之间不共享元数据的访问(即不同 MDS 处理的元数据请求没有交集),但是 MDS 可以访问所有的 OSD 设备.当一个 MDS 失效时,其他 MDS 可以处理该 MDS 负责的元数据,增强容错能力.

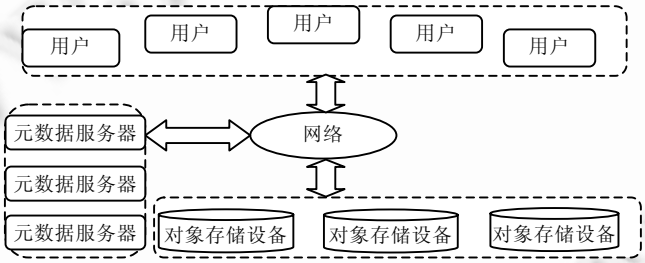


Fig.1 Architecture of metadata server cluster which shares OSD devices

图 1 共享 OSD 设备的元数据服务器集群的体系结构

由于元数据负载的异构特性以及服务器性能的异构性,问题转化为如何在异构的服务器上分布异构的元数据负载,同时保证负载按照服务器的性能公平地分布.下面我们给出问题的形式化定义.

定义 1(元数据服务器集合). 设存储系统中元数据服务器集合 S 为 $\{s_1,\ldots,s_n\}$,其中,n 为服务器的总数,s_i 表示一个设备元素.

定义 2(服务器的异构性). 服务器的异构性体现在计算能力的差异上.设服务器 $s_i \in S$ 的计算能力为 w_i,所有服务器计算能力的集合为 $W\{w_1,\ldots,w_n\}$.

定义 3(元数据集合). 设存储系统中元数据集合为 $X_0=\{x_1,\ldots,x_m\}$,其中,m 表示元数据的总数,$x_i \in X_0$ 表示一个元数据.

定义 4(元数据负载). 元数据的负载随着用户访问次数的变化而变化.设在某个时刻,元数据 $x_i \in X_0$ 的负载大小为 L_i,与所有元数据负载相比,其相对负载大小为 l_i.

定义 5(负载布局函数). 负载布局函数 $f_0: X_0 \to D_0$ 将数据集合 X_0 按照其负载大小映射到异构的设备集合 D_0.

定义 6(负载均衡). 设存储系统的元数据服务器集合为 $S\{s_1,\ldots,s_n\}$,元数据集合为 $X_0\{x_1,\ldots,x_m\}$,负载布局函

数 f_0:$X_0 \to D_0$ 将元数据集合 X_0 映射到服务器集合 S.

若 $\forall i,t \in D_0, \varepsilon > 0$: $\left| \frac{\sum_{j \in X_0 \wedge f(j)=i} l_j}{w_i} - \frac{\sum_{j \in X_0 \wedge f(j)=t} l_j}{w_t} \right| < \varepsilon$, 则函数 f_0 是公平的.

元数据负载均衡的目标是 minimize $\sum_{i \in D_o} \left| w_i - \frac{\sum_{j \in X_0 \wedge f(j)=i} l_j}{\sum_{j \in X_0} l_j} \right|$.

元数据负载均衡问题(metadata load balancing problem,简称 MLBP)是一个单目标优化问题.

3 自适应的分布式负载均衡机制 ADMLB

我们首先假设所有元数据的负载大小是相同的,例如每个元数据有一次访问,而元数据服务器的计算能力是异构的.元数据服务器初始的权值为服务器的计算能力.算法根据服务器的权值公平地分布大小相同的负载.然后,通过调整元数据服务器的权值来进一步体现元数据负载的异构性.这个分解步骤与对象存储系统的实际运行情况是相符的.系统刚配置成功时,上层应用写文件时由 MDS 生成文件的元数据.初始情况下,元数据的负载大小都是相同的,因而我们的前提假设在实际系统刚运行时是成立的.在运行一段时间后,元数据的负载发生变化,我们需要对以前的元数据负载分布进行增量调整,从而适应元数据的负载变化,重新在服务器之间进行负载均衡.因而第 2 步调整服务器的权值,体现异构的元数据负载特征,与实际系统的运行情况是相符的.

我们使用一段时间内的平均访问延迟来评估服务器的负载分布.若服务器的访问延迟相同,则服务器之间的元数据负载分布是均衡的.

我们使用一种自适应的分布式元数据负载均衡机制(ADMLB)来定时地分布服务器之间的元数据负载.该机制在未知元数据负载大小的情况下,使用基本的负载均衡算法(basic load balancing algorithm,简称 BBLA)分配负载.然后,使用增量式的负载均衡算法(incremental load balancing algorithm,简称 IBLA)重新进行负载均衡.

3.1 基本的负载均衡算法

基本的负载均衡算法(BBLA)基于文献[18]中的对数方法,将大小相同的元数据负载分配到异构的 MDS 上.具体来说,对于 $S\{s_1,\ldots,s_n\}$ 中的服务器 s_i,使用函数 h_1:$S \to [0,1]$ 将 $s_i \in S$ 映射到[0,1]环上的某个点,然后根据 h_2:$X \to [0,1]$ 计算 $X_0\{x_1,\ldots,x_m\}$ 中数据元素 $x_j \in X$ 的 hash 值.按照距离函数:

$$d(x_j,s_i): \frac{-(\ln(1-|h_1(x_j)-h_2(s_i)|))}{w_i} \tag{1}$$

将 x_j 分配给离其最近的服务器.初始情况下,所有元数据的负载相同.该距离函数引入了权值 w_i.w_i 初始设置为服务器的相对计算能力,计算能力强的服务器可以被优先考虑,从而才有可能获得更多的元数据.初始情况下,基本的负载均衡算法 BBLA 如图 2 所示.

在该算法中,由于距离函数使用了权值即服务器的相对计算能力,因而负载按照其计算能力公平地分布在各个服务器上.定理 1 证明了服务器上的元数据负载与其计算能力成正比.

定理 1. 基本的负载均衡算法使得元数据负载按照服务器的计算能力均衡地分布在服务器上,即某个服务器上的元数据负载与其计算能力大小成正比.

证明:设系统中元数据服务器集合为 $S\{s_1,\ldots,s_n\}$,其相对计算能力分布为$(w_1,\ldots,w_n)$.元数据集合为 $X_0\{x_1,\ldots,x_m\}$,每个元数据负载大小是相同的,均为 l,则总的元数据负载量为 m.

文献[18]的定理 6 指出:对数方法中,一个数据元素分配给某个节点的概率等于节点的相对权重.在基本的负载均衡算法中,一个元数据负载类似于一个数据元素,元数据服务器相当于节点,因而一个元数据负载分配给服务器的概率等于其相对的计算能力.服务器 s_i 的相对计算能力为 w_i,则一个元数据负载属于服务器 s_i 的概率为 w_i.因为元数据负载量为 m,则服务器 s_i 上期望的元数据负载为 $m \cdot w_i$,因而元数据的负载按照服务器的计算能

力公平地分布.

BBLA (x_j,$S\{s_1,\ldots,s_n\}$,$W\{w_1,\ldots,w_n\}$)
{
For 每个元数据服务器 $s_i \in S$
If $\left(\frac{-(\ln(1-|h_1(x_j)-h_2(s_i)|))}{w_i} < nearest\right)$ then
//*nearest* 用来保存 x_j 与离其最近的服务器之间的距离
$nearest = \frac{-(\ln(1-|h_1(x_j)-h_2(s_i)|))}{w_i}$;
//s 用来保存离 x_j 最近的 s_i
$s=s_i$;
End If
End For
Return s;
}

Fig.2 Basic load balancing algorithm

图 2 基本的负载均衡算法

3.2 分布式的增量负载均衡算法

随着时间的进一步扩展,元数据的负载大小发生了变化.服务器的负载均衡状态被破坏,有些服务器上的负载变大,有些服务器上的负载变小.此时,需要使用增量调整方法,使负载能够重新公平地分布在服务器之间.我们使用访问延迟来表示衡量某个服务器上的负载情况.因为元数据的访问请求是小文件 I/O 操作,每个请求的服务时间的差异性不大,因而使用访问延迟作为衡量指标[19].对于序列 I/O 操作,则使用吞吐量比较合理[19].对于某个时刻 t,观测某个服务器 $s_i \in S$ 上的访问延迟为 $Latency'_i(t)$.使用一个基于权重的移动平均来计算平均访问延迟,避免某个时刻的访问延迟突然变得很大的情况.

$$Latency_i(t) = (1-\alpha)\cdot Latency'_i(t) + \alpha \cdot Latency_i(t-1) \tag{2}$$

使用 FAST TCP 里的控制函数:

$$w_i(t) = (1-\beta)\cdot w_i(t-1) + \beta \cdot \left(\frac{ALatency(t)}{Latency_i(t)}\cdot w_i(t-1) + w_i\right) \tag{3}$$

在公式(3)中, $ALatency(t) = \sum_{i=1}^{n} w_i(t-1)\cdot Latency_i(t)$.服务器 s_i 在时刻 t 的计算能力调整为 $w_i(t)$,按照距离函数(1)重新计算元数据与服务器的距离,将 x_j 分配给离其最近的设备.

这个过程可以采用分布式的方式,在每个服务器计算完新的权值后,广播给其余的服务器,则每个服务器都有所有服务器的新配置信息集合,即$\{w_1(t),w_2(t),\ldots,w_n(t)\}$.对于某个服务器负责的元数据 x_j,使用距离函数(1)计算它与离它最近的服务器的距离,然后通知该服务器负责元数据 x_j 的访问,作为元数据 x_j 的授权服务器.分布式的增量负载均衡算法如图 3 所示.该算法作为一个定时进程运行在每个服务器上,服务器的该进程之间可以相互进行通信.

每个服务器上运行的分布式增量负载均衡算法定时地根据访问延迟调整服务器上的元数据负载.设在时刻 t–1 到时刻 t 这段时间内,服务器 s_i 上的平均访问延迟为 $Latency_i(t)$,所有服务器上的加权平均访问延迟为

$$ALatency(t) = \sum_{i=1}^{n} w_i(t-1)\cdot Latency_i(t).$$

当 $Latency_i(t) \prec ALatency(t)$时,服务器 s_i 应该承担更多的元数据负载.根据公式(3)可知,在时刻 t,服务器 s_i 的计算能力 $w_i(t)$增大.根据定理 1 可知,服务器 s_i 上的负载随着 $w_i(t)$的增大而增加.同理,当 $Latency_i(t) \succ ALatency(t)$时,服务器 s_i 应该减少其负责的元数据负载.根据公式(3)可知,在时刻 t,服务器 s_i 的计算能力 $w_i(t)$减小;由定理 1 可知,服务器 s_i 上的负载随着 $w_i(t)$的减小而降低.因而分布式的增量均衡算法可以动态地调整服务器的负载,使

得负载较轻的服务器负责更多的负载,负载较重的服务器将负载转移到其他服务器上.

IBLA ($x_i \in X, S\{s_1,\ldots,s_n\}, W\{w_1,\ldots,w_n\}$)
{
//收集本服务器上访问延迟的观测值 $Latency_i'$
重置 $Latency_i'$;
//计算该段时间内访问延迟的平均值
$Latency_i = (1-\alpha)\cdot Latency_i' + \alpha \cdot Latency_i$;
向其他服务器发送平均访问延迟;
//收集其他服务器上该段时间内的平均访问延迟$\{Latency_1,\ldots,Latency_{i-1},Latency_{i+1},\ldots,Latency_n\}$
重置$\{Latency_1,\ldots,Latency_{i-1},Latency_{i+1},\ldots,Latency_n\}$;
//计算所有服务器的加权平均访问延迟 $ALatency$
$ALatency = \sum_{i=1}^{n} w_i \cdot Latency_i$;
//使用控制函数调整服务器 s_i 的权值,其调整的权值表示为 w_i'
$w_i' = (1-\beta)\cdot w_i' + \beta \cdot \left(\frac{ALatency}{Latency_i}\cdot w_i' + w_i\right)$;
//收集其他服务器的最新权值
重置其他服务器的权值集合 $\{w_1',...,w_{i-1}',w_{i+1}',...,w_n'\}$;
For 服务器 s_i 负责的每个数据 $x_j \in X_i$
//按照基本的负载均衡算法重新计算负责 x_j 的元数据服务器
s=BBLA ($x_j, S\{s_1,\ldots,s_n\}, \{w_1',...,w_n'\}$);
If (s!=s_i) then 将元数据 x_j 的负载交给服务器 s 来处理;
End If
End For
}

Fig.3 Distributed incremental load balancing algorithm

图 3 分布式的增量负载均衡算法

4 ADMLB 机制的特性分析

在第 3 节里,我们讨论了 ADMLB 机制的负载均衡性.本节主要从 ADMLB 机制的自适应性、容错特性以及元数据服务器的快速定位这 3 个方面来进行分析.

4.1 自适应性

4.1.1 元数据的增加和删除

ADMLB 机制可以适应元数据的增加和删除,同时重新保证元数据负载的均衡分布.存储系统的用户首先获取所有 MDS 的相对计算能力分布信息$\{w_1,w_2,\ldots,w_n\}$,将其设置为服务器的初始权值.对于要访问的文件元数据,使用距离函数 $d(x_j,s_i)$:$\frac{|h_1(x_j)-h_2(s_i)|}{w_i}$ 计算元数据与服务器的距离,然后选择距离最近的元数据服务器,向其发送元数据的读写请求.对于某个元数据的增加或者删除,使用类似的方法判断负责该元数据的服务器,然后向服务器发送请求,由元数据服务器与底层的设备进行通信,执行增加或者删除元数据的操作.为了保证元数据负载的均衡分布,ADMLB 定时地根据服务器的访问延迟调整其计算能力,增大访问延迟小的服务器的权值,减小访问延迟大的服务器的权值,然后向其他服务器广播其新的权值配置信息.采取分布式的方式来调整各个服务器上的负载,最终使元数据服务器的访问延迟能够与其计算能力成正比.

4.1.2 元数据服务器的增加和删除

大规模对象存储系统的建成不是一蹴而就的,随着时间的推移以及数据量的增加,不断增加新设备.由于访问文件之前要访问元数据,如果元数据服务器的性能滞后,则必然导致底层设备的功能无法发挥,从而使得增加新设备没有任何意义,因而需要增加元数据服务器来满足大量数据的访问需求.设系统的元数据服务器集合为

$S\{s_1,\ldots,s_n\}$,增加新服务器 s_{n+1} 后,每个服务器的权值发生变化,按照距离函数重新计算元数据的授权服务器.在元数据服务器增加时,ADMLB 机制只需要修改服务器权值的配置信息,重新分配元数据的授权服务器.同理,当删除旧的服务器 $s_i \subseteq S$ 时,修改剩余服务器的权值,将服务器 s_i 负责的元数据重新分布到其他服务器上.

4.1.3 元数据访问热度

在初始情况下,每个元数据的负载大小是相同的,即每个元数据的访问热度相同.使用基本的负载均衡算法使负载公平分布到各个服务器上.当元数据的访问热度变化(即负载变化)时,每个服务器上的访问延迟可以将这种变化体现出来.当某个服务器的访问延迟低于平均访问延迟时,表示该服务器上的元数据访问热度高;当某个服务器的访问延迟低于平均访问延迟时,表示该服务器上的元数据访问热度低.通过本文提出的分布式的增量负载均衡算法,可以动态地调整每个服务器上负载的分布,最终使得每个服务器的平均访问延迟相同,从而使得元数据负载再次均衡分布,解决了某些元数据访问较热或者较冷的问题.

4.2 容错特性

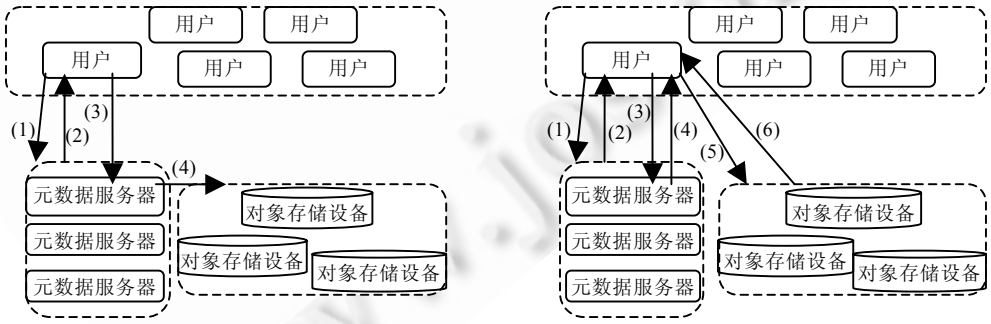
当一个服务器失效时,该服务器负责的元数据负载将会由其他服务器来承担.服务器失效类似于服务器的删除,被删除的服务器在离开系统之前可以向其他服务器广播删除行为,从而使其他服务器能够对现有的计算能力分布进行修改.但是失效服务器无法预测失效行为,不可能告知其他服务器的失效行为.此时,检测服务器失效的方法是:当用户访问某个服务器 s_i 时,发现其为失效状态,则随机地选择一个服务器 $s_j(j\neq i)$发送请求,同时告知服务器 s_i 的失效行为.服务器 $s_j(j\neq i)$将 s_i 已失效通知其他服务器.每个服务器修改计算能力的配置信息以及调整负载,将失效服务器的负载分摊到其他服务器上.当系统重新达到平衡时,失效服务器上的负载可以公平地分布到其他服务器上.因而,ADMLB 机制具有很好的容错特性.

4.3 元数据服务器的定位及访问流程

随着时间的变化,各个元数据服务器的计算能力也会发生变化.使用存储系统的用户需要定时地获取每个元数据服务器的计算能力,从而按照第 2 节描述的负载均衡机制来高效地定位服务器.读写数据的元数据时,首先要判断负责该元数据的服务器,即定位授权服务器.在首次使用存储系统时,先从各个服务器获取实时的配置信息,只需了解服务器最新的计算能力即可.然后,定时地获取服务器关于计算能力的配置信息.

在访问数据 x_j 的元数据时,向 $\min_i \frac{-(\ln(1-|h_1(x_j)-h_2(s_i)|))}{w_i}$ 值对应的元数据服务器 s_i 发送元数据服务器的访问请求.计算 $\min_i \frac{-(\ln(1-|h_1(x_j)-h_2(s_i)|))}{w_i}$ 的时间复杂度为 $O(\log n)$,其中,n 为元数据服务器的个数.

用户写元数据的流程如图 4 所示.



(a) 写元数据的流程图 (b) 读元数据的流程图

Fig.4 Flow chart of reading and writing metadata

图 4 读写元数据的流程图

(1) 用户在首次使用系统时,向所有的 MDS $S\{s_1,\ldots,s_n\}$发送请求,获取其关于当前计算能力的配置信息,

将其作为MDS的初始权值.在系统刚投入运行时,MDS的计算能力等于其硬件的配置.系统运行一段时间内,MDS的权值等于按照公式(3)调整后的值.

(2) MDS向用户返回新权值的配置信息,则用户拥有所有MDS的权值最新配置信息$\{w_1,\ldots,w_n\}$.

(3) 用户需要写某个数据x_i时,使用公式(1)计算单位环上离数据x_i最近的MDS,然后向MDS发送写数据的请求.

(4) MDS接收用户的请求后,生成数据x_i的元数据,向底层的OSD写元数据信息.

读元数据的流程与写元数据的流程相似,区别在于用户在读元数据时可以直接与OSD设备进行通信.修改写流程中的第(4)步,分成以下3步(如图4中第(4)步~第(6)步所示,前第(1)步~第(3)步与写元数据的步骤相同):

(4) MDS接收用户的请求后,计算数据x_i的元数据所在的OSD设备,将结果返回给用户.

(5) 用户根据返回的结果直接向OSD设备发送数据x_i的元数据读请求.

(6) OSD设备将数据x_i的元数据返回给用户.

负载均衡算法定时地调整负载的分布,可以在系统空闲的情况下调用分布式的增量负载均衡算法.在获取服务器的新权值配置信息之后,定位到元数据访问所在的服务器,用户与服务器按照正常的流程读写元数据,不会影响读写元数据本身带来的开销.

5 实验与结果分析

为了测试ADMLB算法的可行性,本文在模拟环境下实现了ADMLB,与随机的方法RM以及基本的BBLA方法进行比较分析.算法在一台PC机上实现.该PC机拥有3.0GHz Pentium 4 CPU以及1.0GB内存,运行Windows XP操作系统和Eclipse-SDK-3.3.1.1-win32.我们使用人工合成的负载以及负载大小,设系统中有1 000个元数据,初始状态下,每个元数据的负载大小相同,设为1.然后,随机地生成每个元数据的负载,值在0~10之间.我们模拟5个元数据服务器,其性能比为1:2:3:4:5,则在相同的时间段内,每个MDS完成的负载比为1:2:3:4:5.即在某个时间段内,若第1个MDS完成1次元数据的访问,那么第5个MDS可以完成5次元数据的访问.设每个MDS完成一次元数据访问的时间分别为0.25ms,0.25/2ms,0.25/3ms,0.25/4ms和0.25/5ms.使用ADMLB算法将1 000个负载分配到各个元数据MDS上.假设对元数据的每个访问都是在同一时间达到的.假设每隔60ms调用一次ADMLB算法,负载的到达时间在0~60ms之间,随机地模拟负载的到达时间.统计最近60ms之内各个MDS的平均访问延迟,直到每个MDS的平均访问延迟相同为止.取α=0.2来计算平均访问延迟.因为负载大小的异构性以及元数据服务器的异构性,负载的分布不可能达到绝对的平衡.例如,系统中有3个MDS,性能比为1:2:3,3个元数据的负载分别为1,1和3.在这种情况下,不可能将负载绝对公平地分布到各个MDS上,那么算法会进入死循环状态,永远无法收敛.为了避免这种情况,我们设定一个参数γ,值在0~1之间.当每个MDS的访问延迟在平均访问延迟的$(1-\gamma)$倍时,算法停止.初始情况下,取γ=0.2,β=0.2,比较3种算法的访问延迟.图5~图7显示了ADMLB,RM以及BBLA这3种方法的访问延迟情况.

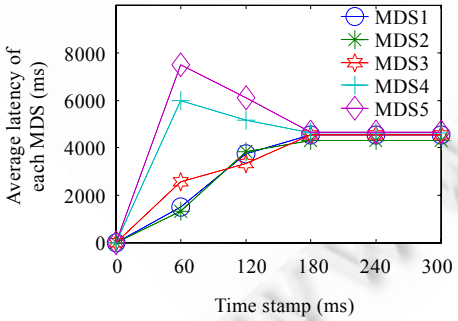


Fig.5 Latency of ADMLB

图5 ADMLB的访问延迟

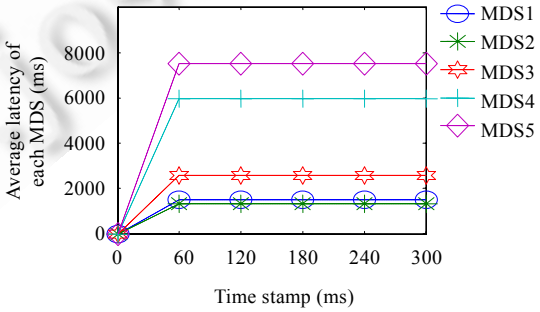


Fig.6 Latency of BBLA

图6 BBLA的访问延迟

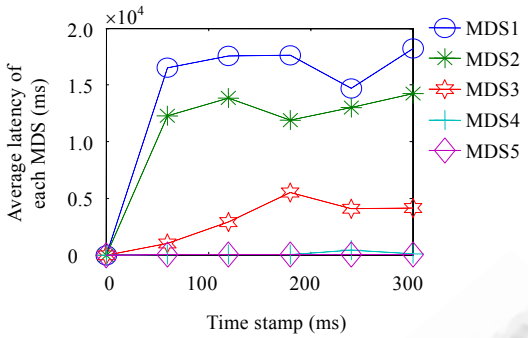


Fig.7 Latency of RM

图 7 RM 的访问延迟

初始时刻,ADMLB 与 BBLA 的访问延迟相同,随着时间的推移,ADMLB 不断地对各个 MDS 上的负载进行调整,直到最后达到稳定状态.BBLA 在初始情况下根据 MDS 的性能分布负载,不能根据当时的访问延迟调整负载的分布,因而访问延迟没有变化.RM 不能根据访问延迟以及 MDS 性能来调整负载的分布,因此不能调用 RM 使得负载达到均衡.经过一段时间的负载均衡,当每个 MDS 的访问延迟达到标准时,ADMLB 算法终止.图 8 显示了在第 300s 时,3 种算法的每个 MDS 上的负载大小之和与总负载之比.使用 ADMLB 以及 BBLA 算法,每个 MDS 上的负载大小之和与总负载接近于每个 MDS 的性能所占的比例;而使用 RM 算法时,每个 MDS 的负载大小之和几乎相同,没有按照 MDS 的性能进行均衡分布.

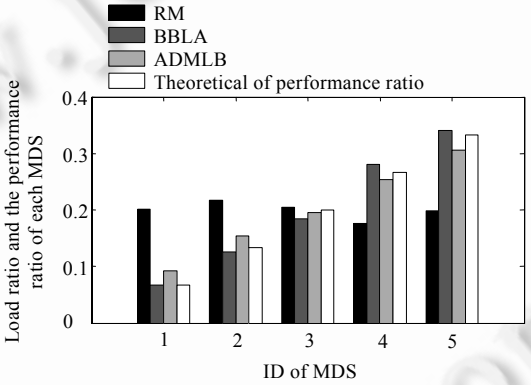


Fig.8 Load ratio and the performance ratio of each MDS

图 8 每个 MDS 的负载比例和性能比例

6 总 结

面向对象的存储系统可以提供海量信息的存储,提供设备级的安全性,广泛应用于科学计算以及其他数据密集型应用.在对象存储系统中,访问数据之前必须要访问元数据.元数据访问很容易成为性能瓶颈.因而,合理的元数据管理,对于提高整个系统的 I/O 性能具有重要的意义.如何在元数据服务器之间均衡地分布元数据负载,是元数据管理的关键技术之一.现有的元数据负载均衡机制采用的是集中式的方法.本文设计了一种分布式的元数据负载均衡机制,可以按照服务器的性能来分布负载;同时,采取分布式的方式定时地调整服务器之间动态变化的负载.不仅能够自适应负载的变化,而且能够自适应服务器规模的变化.在服务器增加和删除时,动态地均衡负载.在服务器失效时,采用有效的机制将负载分摊到其他服务器上.在保证负载均衡分布、自适应以及容错等特性的同时,可以使用预定义的函数快速地定位元数据服务器.

References:

[1] Sun Microsystems, Inc. Lustre file system: High-Performance storage architecture and scalable cluster file system. 2007. https://www.sun.com/offers/docs/LustreFileSystem.pdf

[2] Pawlowski B, Juszczak C, Staubach P, Smith C, Lebel D, Hitz D. NFS version 3: Design and implementation. In: Proc. of the USENIX Summer 1994 Technical Conf. San Jose: USENIX, 1994. 137−152.

[3] Morris JH, Satyanarayanan M, Conner MH, Howard JH, Rosenthal DS, Smith FD. Andrew: A distributed personal computing environment. Communications of the ACM, 1986,29(3):184−201. [doi: 10.1145/5666.5671]

[4] Satyanarayanan M, Kistler JJ, Kumar P, Okasaki ME, Siegel EH, Steere DC. Coda: A highly available file system for a distributed workstation environment. IEEE Trans. on Computers, 1990,39(4):447−459. [doi: 10.1109/12.54838]

[5] Weil SA, Pollack KT, Brandt SA, Miller EL. Dynamic metadata management for petabyte-scale file systems. In: Proc. of the 2004 ACM/IEEE Conf. on Supercomputing. Washington: IEEE Computer Society, 2004. [doi: 10.1109/SC.2004.22]

[6] Wu JJ, Liu PF, Chung YC. Metadata partitioning for large-scale distributed storage systems. In: Proc. of the 3rd Int'l Conf. on Cloud Computing. Washington: IEEE Computer Society, 2010. 212−219. [doi: 10.1109/CLOUD.2010.24]

[7] Xiong J, Hu Y, Li G, Tang R, Fan Z. Metadata distribution and consistency techniques for large-scale cluster file systems. IEEE Trans. on Parallel and Distributed Systems, 2011,22(5):803−816. [doi: 10.1109/TPDS.2010.154]

[8] Brandt SA, Miller EL, Long DDE, Xue L. Efficient metadata management in large distributed storage systems. In: Proc. of the 20th IEEE/the 11th NASA Goddard Conf. on Mass Storage Systems and Technologies. Washington: IEEE Computer Society, 2003. 290−298. [doi: 10.1109/MASS.2003.1194865]

[9] Liu Z, Zhou XM. A metadata management method based on directory path. Ruanjian Xuebao/Journal of Software, 2007,18(2): 236−245 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/236.htm [doi: 10.1360/jos180236]

[10] Li WJ, Xue W, Shu JW, Zheng WM. Dynamic hashing: Adaptive metadata management for petabyte-scale file systems. In: Proc. of the 23rd IEEE/the 14th NASA Goddard Conf. on Mass Storage System and Technologies (MSST 2006). Washington: IEEE Computer Society, 2006.

[11] Zhu Y, Jiang H, Wang J. HBA: Distributed metadata management for large cluster-based storage systems. IEEE Trans. on Parallel and Distributed Systems, 2008,19(4):1−14. [doi: 10.1109/TPDS.2008.33]

[12] Hua Y, Zhu YF, Jiang H, Feng D. Scalable and adaptive metadata management in ultra large-scale file systems. In: Proc. of the 28th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS 2008). Washington: IEEE Computer Society, 2008. 401−408. [doi: 10.1109/ICDCS.2008.32]

[13] Guo CC, Yan PL. A dynamic load balancing algorithm for heterogeneous Web server cluster. Chinese Journal of Computers, 2005, 28(2):179−184 (in Chinese with English abstract).

[14] Ni YZ, Lu GH, Huang YH. The solution of disk load balancing based on disk striping with genetic algorithm. Chinese Journal of Computers, 2006,29(11):1995−2002 (in Chinese with English abstract).

[15] Chen T, Xiao N, Liu F, Fu CS. Clustering-Based and consistent hashing-aware data placement algorithm. Ruanjian Xuebao/Journal of Software, 2010,21(12):3175−3185 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3706.htm [doi: 10.3724/SP.J.1001.2010.03706]

[16] Shan ZG, Dai QH, Lin C, Yang Y. Integrated schemes of Web request dispatching and selecting and their performance analysis. Ruanjian Xuebao/Journal of Software, 2001,12(3):355−366 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/12/355.htm

[17] Jin C, Wei D, Low S. FAST TCP: Motivation, architecture, algorithms, performance. In: Proc. of the 23rd Conf. of the IEEE Communications Society (INFOCOM 2004). Washington: IEEE Computer Society, 2004. [doi: 10.1109%2fINFCOM.2004.1354670]

[18] Schindelhauer C, Schomaker G. Weighted distributed hash tables. In: Paul S, ed. Proc. of the 17th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA 2005). Las Vegas: ACM Press, 2005. 218−227. [doi: 10.1145/1073970.1074008]

[19] Wu CX, Burns R. Handling heterogeneity in shared-disk file systems. In: Proc. of the 2003 ACM/IEEE Conf. on Super Computing (SC 2003). Washington: IEEE Computer Society, 2003. [doi: 10.1145/1048935.1050158]

附中文参考文献:

[9] 刘仲,周兴铭.基于目录路径的元数据管理方法.软件学报,2007,18(2):236−245. http://www.jos.org.cn/1000-9825/18/236.htm [doi: 10.1360/jos180236]

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

[14] 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.计算机学报,2006,29(11):1995−2002.

[15] 陈涛,肖侬,刘芳,付长胜.基于聚类和一致 Hash 的数据布局算法.软件学报,2010,21(12):3175−3185. http://www.jos.org.cn/1000-9825/3706.htm [doi: 10.3724/SP.J.1001.2010.03706]

[16] 单志广,戴琼海,林闯,杨扬.Web 请求分配和选择的综合方案与性能分析.软件学报,2001,12(3):355−366. http://www.jos.org.cn/1000-9825/12/355.htm

陈涛(1982－),女,湖北荆州人,博士,助理研究员,主要研究领域为网络存储,数据库,质谱数据分析.
E-mail: lovely696521@163.com

刘芳(1976－),女,博士,副教授,主要研究领域为信息安全,网络存储.
E-mail: liufang@nudt.edu.cn

肖侬(1969－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为分布计算,网格计算,网络存储.
E-mail: xiao-n@vip.sina.com