而不考虑服务器本身的性能差异.这样就可能出现高性能服务器负载虽然超过了集群负载的阈值,但服务器本身其实没有超载,而文件系统依然会启用负载均衡机制将负载迁移到了集群中性能较弱的服务器上,这会导致一些不必要的资源开销,而且很有可能出现性能较弱的服务器在完成子树迁移之后负载过高,服务器压力过大,又要再次进行迁移,此时就会造成系统抖动,极大地延长了负载均衡的时间.所以为了避免这种情况,本文充分考虑到了元数据服务器的性能差异.我们把从上一节获得的性能值 P 用于表示该元数据服务器处理元数据请求的能力.每个元数据服务器在元数据服务器集群中用 UDP 广播包广播自己的性能值、负载量、负载信息和时延量,并由负载水平最低的元数据服务器作为决策元数据服务器.决策元数据服务器记录发送的时间、负载均衡的周期等信息.决策元数据服务器收集集群内所有服务器的性能值 $P=[p_1,p_2,p_3,\ldots,p_n]$以及所有服务器的负载量 $Load=[load_1,load_2,load_3,\ldots,load_n]$,然后计算出集群的总性能值 P_{total} 和总负载量 $Load_{\text{total}}$,见公式(17)、公式(18).

$$P_{\text{total}}=\sum_{k=1}^{n}P_k \tag{17}$$

$$Load_{\text{total}}=\sum_{k=1}^{n}Load_k \tag{18}$$

在异构的元数据服务器中,服务器的性能越强,其应该负担的负载量越大,这样才可以充分利用集群的系统资源,又可以平衡各个元数据服务器的处理效率.通过上述的总性能值和总负载值,我们可以得到第 i 号元数据服务器在负载均衡的情况下的目标负载量 $Load_{target(i)}$,见公式(19).

$$Load_{target(i)}=\frac{P_i}{P_{\text{total}}}\times Load_{\text{total}} \tag{19}$$

接下来可以计算出第 i 号元数据服务器的现有负载值与其目标负载值的差距$\Delta Load$,见公式(20).

$$\Delta Load=Load_i-Load_{target(i)} \tag{20}$$

如果$\Delta Load>0$,表示该元数据服务器的负载值相对过高,需要向其他元数据服务器迁移目录子树;如果$\Delta Load<0$,表示该元数据服务器的负载值相对过低,可以接收其他元数据服务器迁移过来的目录子树.同时,我们设置一个触发阈值$\Delta Load_{\text{thredhold}}$,如果该周期内有元数据服务器的$|\Delta Load|>\Delta Load_{\text{thredhold}}$,则触发目录子树迁移;否则不触发.设置阈值的目的是防止迁移过于频繁而导致的系统性能抖动或消耗增加.

设置 $EXSET$,$INSET$ 分别为迁出子树元数据服务器和迁入子树元数据服务器的集合,如果元数据服务器的$\Delta Load>0$,且$|\Delta Load|>\Delta Load_{\text{thredhold}}$,则将其加入 $EXSET$;如果$\Delta Load<0$,且$|\Delta Load|>\Delta Load_{\text{thredhold}}$,则加入 $INSET$.自适应负载均衡算法如算法 1 所示.

算法 **1**. 自适应负载均衡算法.

Task T1: A self-adaption process judgement

```
1.  while (true)
2.    R_change=R_before−R; R_before=R;
3.    if R_change>R_thredhold then
4.      break;
5.    else
6.      sleep 5 seconds;
7.    endif
8.  endwhile
```

Task T2: A self-adaption process initiation

```
9.    Req=getReq();
10.   Lag=gerLag();
11.   Queue.push(⟨Req,Lag⟩); Queue.pop();
```

12. using data in *Queue* to calculate *Slope*;
13. $P=a/Slope$;
14. $Load=H+w\times L$;
15. *broadcast*(*Load*,*P*);

Task T3: The MDS start to make load-balancing

16. $Load_{total}=0$; $P_{total}=0$;
17. **for each** MDS *i* **do**
18. $calarry.add(\langle Load_i,p_i\rangle)$;
19. $Load_{total}=Load_{total}+Load_i$;
20. $P_{total}=P_{total}+P_i$;
21. **endfor**
22. **for each** MDS *i* **do**
23. $Load_{target(i)}=P_i/P_{total}\times Load_{total}$;
24. **if** $Load_i-Load_{target(i)}>Load_{thredhold}$ **then**
25. put MDS_i into *EXSET*;
26. **endif**
27. **if** $Load_{target(i)}-Load_i>Load_{thredhold}$ **then**
28. put MDS_i into *INSET*;
29. **endif**
30. **endfor**

2.3 目录子树迁移过程

当决策元数据服务器选出负载水平较大的节点和负载水平较小的节点后,需要进行目录子树迁移.目录子树迁移[22]主要包括以下 3 个过程:首先进行一次负载信息收集工作,由决策元数据服务器匹配迁出子树的元数据服务器和迁入子树的元数据服务器及其迁移量;接下来,根据迁移量的大小,在迁出子树的元数据服务器上执行拆分子树的操作;最后,将拆分完成的子树迁移到迁入子树的元数据服务器中.

(1) 元数据服务器匹配

首先,将集合 *EXSET* 和 *INSET* 中的元数据服务器按照其$|\Delta Load|$大小做降序排列;接下来,将 *EXSET* 中的第 *i* 位与 *INSET* 中的第 *i* 位做匹配,*i* 从 1 开始,其中,*EXSET* 的第 *i* 位作为目录子树的迁出方,*INSET* 的第 *i* 位作为目录子树的接收方,直到两个集合中有一个集合的所有元数据服务器都匹配完成为止.其中,*EXSET* 的第 *i* 位向 *INSET* 的第 *i* 位迁移时,以两者的$|\Delta Load|$的较小值作为迁入迁出的负载量 $Load_m$.

(2) 子树拆分

在确定了外迁的负载量 $Load_m$ 之后,需要迁出目录子树的元数据服务器需要将负载值总和为 $Load_m$ 的目录子树拆分出来,为迁移做准备.

首先,设 *S* 和 *M* 分别为放置负载量小于 $Load_m$ 的目录子树的集合以及准备进行迁移的目录子树的集合;同时,设置一个阈值 $Load_t$ 来表示目录负载与 $Load_m$ 的误差允许范围;接下来,设 *T* 用于放置从目录树根节点出发的所有目录子树.在 *T* 中选取一个目录子树 $Tree_j$,计算其负载量 $Load_j$,其中,$Load_j$ 是指子树的所有节点负载的和.用目录子树 $Tree_j$ 的负载量 $Load_j$ 与迁入迁出的目标负载量 $Load_m$ 对比:如果它们的差在误差 $Load_t$ 范围内,则将该目录子树添加进 *M*,准备外迁;如果它们的差超出了误差 $Load_t$ 范围,而且 $Load_j$ 小于 $Load_m$,则将该目录子树添加进 *S*,然后在 *T* 中继续选取目录子树.子树拆分算法如算法 2 所示.

算法 **2**. 子树拆分算法.

1. **initial** *T*, *S* and *M*;
2. **for** *i* in the size of *T* **do**

3. calculate $Load_j$ of $Tree_j$;
4. remove $Tree_j$ from T;
5. **if** $|Load_j-Load_m|<Load_t$ **then**
6. put $Tree_j$ into M;
7. goto line (30);
8. **else if** $Load_j-Load_m<0$ **then**
9. Put $Tree_j$ into S;
10. **endif**
11. **if** T is empty **then**
12. **break**;
13. **endif**
14. **endfor**
15. Sort S in descending order and set q=0;
16. **for** q in size of S **do**
17. **if** $Load_m\tilde{}Load_q>0$ **then**
18. put S_q into M;
19. $Load_m=Load_m\tilde{}Load_q$;
20. continues;
21. **endif**
22. **if** $Load_m<Load_t$ **then**
23. **break**;
24. **endif**
25. **if** $Load_m\tilde{}Load_q<0$ and $|Load_m-Load_q|<Load_t$ **then**
26. put S_q into M;
27. **break**;
28. **endif**
29. **endfor**
30. choose M as the export set

(3) 目录子树外迁

在确定好要外迁的目录子树之后,迁出节点对该目录子树进行加锁操作,保证在迁移期间中断所有对该目录子树的元数据服务.同时,向集群中所有与该目录子树有逻辑关系的元数据服务器广播通知该目录子树即将被迁移.接下来,对该目录子树进行序列化编码,并发送到迁入节点.迁入节点在接受到目录子树的编码后进行解码,更新到自己的目录上,并且告知迁出节点已经收到迁移的目录子树.同时,广播通知集群中与该目录子树有逻辑关系的元数据服务器迁移完成.最后,迁出节点对该目录子树进行解锁操作,并删除其信息.所有与该目录子树有逻辑关系的元数据服务器更新目录信息,以保证能够正确工作.

3 实 验

本节对新提出的自适应元数据服务负载均衡策略进行验证,实验证明了该策略的可行性、有效性和稳定性.同时,把该策略与 Ceph 负载均衡策略(ceph load balancing strategy,简称 CLBS)作对比,证明了该策略更适用于异构环境,能够更有效地减少集群系统抖动,确保操作时延相对均衡.

3.1 实验环境

本实验在局域网中搭建了一个 Ceph 集群,包括 5 台异构 MDS(元数据服务器)用于文件系统的元数据管

理,1 台 MON(Monitor 服务器)用于检测集群运行情况,还有 1 台 OSD(对象存储服务器)用于文件数据存储.此外,还包括 10 台客户端机组成的一个用户集群.实验环境的配置见表 1.

Table 1 Hardware configuration

表 1 硬件配置

服务器	中央处理器	内存	硬盘	操作系统	带宽
MON	4×Xeon 5580, 2.27Ghz	4G	500G	Ubuntu12.04/kernel3.5.0	100M
OSD	4×Xeon 5580, 2.27Ghz	4G	500G	Ubuntu12.04/kernel3.5.0	100M
MDS1	8×Xeon 5580, 2.67Ghz	8G	500G	Ubuntu12.04/kernel3.5.0	100M
MDS2	4×Xeon 5580, 2.67Ghz	12G	500G	Ubuntu12.04/kernel3.5.0	100M
MDS3	4×Xeon 5650, 2.27Ghz	8G	500G	Ubuntu12.04/kernel3.5.0	100M
MDS4	2×Xeon 5650, 2.27Ghz	8G	500G	Ubuntu12.04/kernel3.5.0	100M
MDS5	2×Xeon 5650, 2.27Ghz	4G	500G	Ubuntu12.04/kernel3.5.0	100M

3.2 实验分析

根据前文提到的性能估算方法,我们对 5 个 MDS 分别进行实验.在保证 MDS 零负载和关闭负载均衡功能的情况下,使用 mdtest 工具分别对每个 MDS 进行压力测试.实验对每个 MDS 逐步线性增加元数据请求量,并记录下每个 MDS 在 8 个标记点的的时延值,记录见表 2,其中,1X 表示 1 倍的元数据请求,依此类推,8X 表示 8 倍的元数据请求.

Table 2 Delay of MDS

表 2 元数据服务器的延迟量

	1X	2X	3X	4X	5X	6X	7X	8X
MDS1	111.4	115.4	118.6	123.2	127	130.2	133.6	140
MDS2	117.2	121.5	125.2	129.8	134.1	139.1	142.5	148.3
MDS3	124.5	130.2	135	139.5	144.1	151.3	155.1	162
MDS4	135.1	141.7	148.2	154.4	161.1	167.7	174.1	180.5
MDS5	141	148.9	167.2	165.1	173	180.8	188.8	197

我们以 MDS1,MDS5 为例,画出元数据请求数量变化与延迟量的关系图,如图 2 所示.

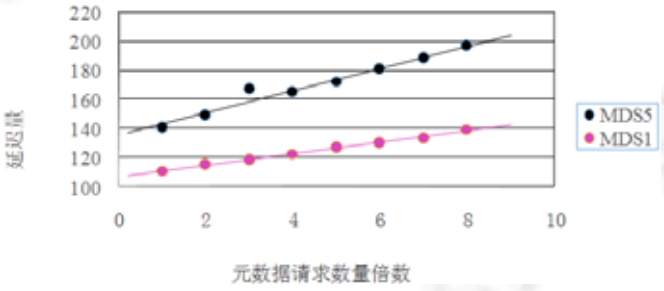

Fig.2 Relation of metadata request quantity and delay

图 2 元数据请求数量变化与延迟量的关系图

从图 2 可以看出,离散点分布明显趋于线性增长的趋势,也就证明了前文在性能估算模型中的结论,在排除其他因素的情况下,元数据请求量与该 MDS 的延迟量成线性关系;同时,我们利用最小二乘法,分别得到这些离散点的线性拟合直线的斜率,然后计算我们的性能估算值 P.我们通过计算这 5 个 MDS 的性能值,从而得到它们的性能比例为 8:6.5:5:4:3.5.

为了验证本文提出的负载均衡策略的优越性,我们采用 3 组实验,分别为关闭负载均衡机制、使用 Ceph 自带的 CLBS 负载均衡策略以及使用本文提出的负载均衡策略这 3 种方法.每组实验运行 10 小时,利用 mdtest 发送元数据请求,每 15 分钟记录下每一台 MDS i 的 CPU 使用率 $Percent_{cpu(i)}$和内存使用率 $Percent_{memory(i)}$.我们用公式(21)来表示每一台 MDS i 的负载情况 L_i:

$$L_i=0.6\times Percent_{cpu(i)}+0.4\times Percent_{memory(i)} \quad (21)$$

我们观察上述 3 组实验中每一台 MDS i 的负载情况 L_i.图 3 为关闭负载均衡机制的情况下,每台 MDS 的负载随时间的变化情况.图 4 为开启 Ceph 自带的负载均衡策略 CLBS 的情况下,每台 MDS 的负载随时间的变化情况.图 5 为使用本文提出的负载均衡策略的情况下,每台 MDS 的负载随时间的变化情况.

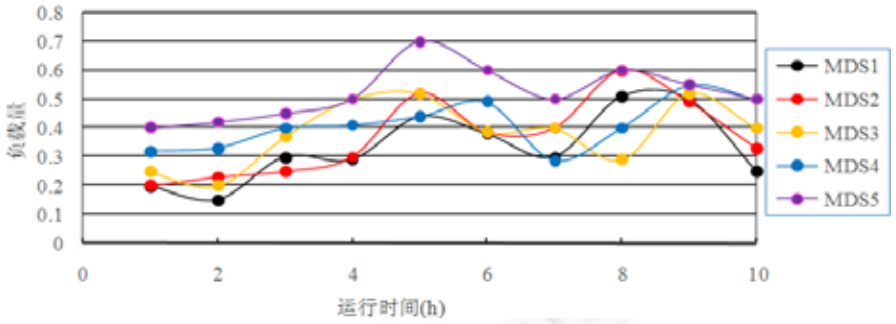

Fig.3 Close the load balancing mechanism

图 3 关闭负载均衡机制的情况

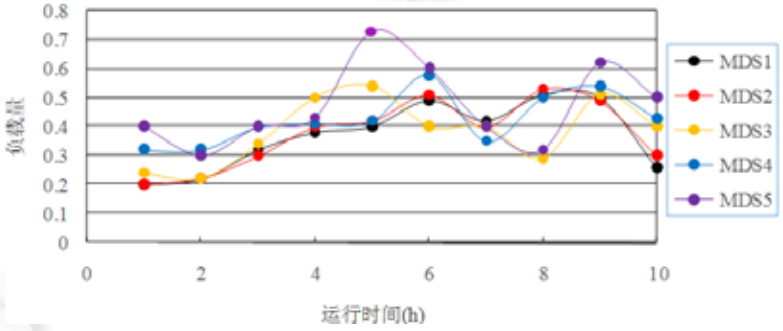

Fig.4 Use the Ceph load balancing strategy

图 4 开启 Ceph 负载均衡策略的情况

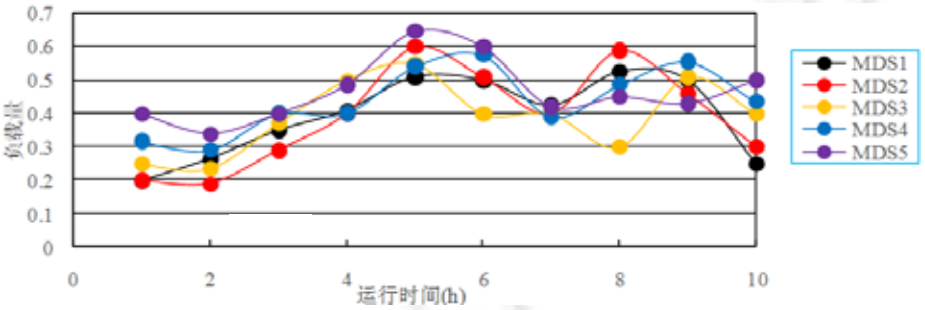

Fig.5 Use load balancing strategy presented in this paper

图 5 开启本文负载均衡策略的情况

从图 3 可以看出,由于访问的随机性,在关闭负载均衡机制的情况下,每一台服务器的负载情况差异都比较大.从图 3 可以看出,MDS5 的性能比较弱,所以它的负载值一直处于偏高状态;而 MDS1 的性能比较强,负载量一直较低,整个硬件使用率也较低.

从图 4 可以看出,使用了 Ceph 自带的 CLBS 负载均衡策略之后,整个负载情况比较接近,差异并不是很大.

但是从图 4 可以看到,在 4 时,MDS1 的负载量逐渐增高后又慢慢下降;而同时,MDS5 的负载却快速升高,然后又回落,所以可以判定这里出现了一次从 MDS1 向 MDS5 的目录子树迁移;然后,MDS5 又将部分目录子树再次迁出,导致系统性能抖动.发生这种情况的主要原因是:CLBS 只考虑到每个 MDS 的热点值对比而没有考虑服务器异构的问题,所以在没考虑到 MDS5 性能较弱的情况下向其迁入热度过多的目录子树,从而导致 MDS5 负载暴增,所以不得已再次进行迁移,本来一次可以实现的子树迁移过程,CLBS 用了两步才完成,这就是我们前文提到的 Ceph 负载均衡策略在异构服务器上的缺点.

从图 5 可以看出,在开启本文提出的负载均衡策略的情况下,每台 MDS 负载量相差不远,波动相对于 CLBS 来讲也小了很多,整个集群表现比 CLBS 更好.实验结果表明,本文提出的方法相对于 Ceph 自带的 CLBS 负载均衡策略来说,在处理异构服务器问题上表现更好.

为了表示整个集群的负载均衡程度,我们计算集群中某一个时刻 MDS 负载量之间的差异性,差异值用 D_L 表示,见公式(22).

$$D_L=\sqrt{\sum_{i=1}^{n}(L_i-\bar{L})^2} \tag{22}$$

用某一个时刻的 MDS 负载量之间的差异值 D_L 来表示该时刻集群的负载均衡程度.接下来,将 10 个时刻的 MDS 负载量的差异值 D_L 相加作为该实验的 MDS 集群的负载均衡程度 D_{total}.D_{total} 的数值越小,表示 MDS 集群的负载程度越好.

图 6 为未开启负载均衡机制的情况下、开启 Ceph 自带的负载均衡策略 CLBS 的情况下和使用本文提出的负载均衡策略的情况下,整个集群的负载差异值的变化.

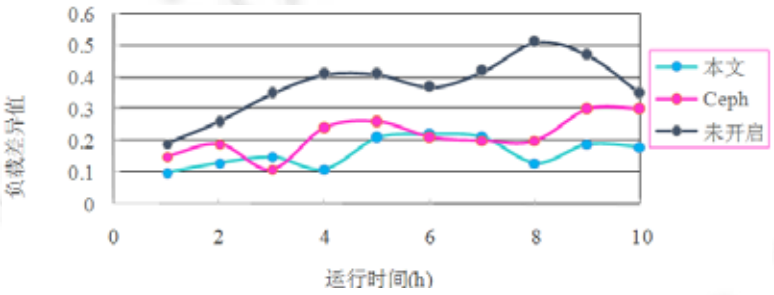

Fig.6 Comparison of variance of three kinds of scheme

图 6 3 种方案的负载差异值对比

从图 6 可以看出,在这 10 个小时中,整个集群的访问量有两次高潮.未开启负载均衡策略的实验中,负载均衡度基本随着访问量的变化而变化;而开启 Ceph 自带的负载均衡策略 CLBS 和使用本文提出的负载均衡策略两种方法都能收敛到一个较好的负载均衡水平.特别是本文的方法,收敛得更快,波动更小.通过计算,可以得到未开启负载均衡机制的实验中,MDS 集群的负载均衡程度 D_{total} 值为 3.2;而开启 Ceph 负载均衡策略 CLBS 的实验中,MDS 集群的负载均衡程度 D_{total} 值为 1.98.相对于第 1 组实验来讲,MDS 集群的负载均衡程度有了明显的改善.使用本文的负载均衡策略的实验中,MDS 集群的负载均衡程度 D_{total} 值为 1.53,相对于 CLBS 方案的实验,MDS 集群的负载均衡程度也有了进一步的改善.

接下来,我们通过实验验证本文提出的方案中的自适应机制的有效性和可用性.本文描述的自适应机制是指系统监控元数据服务器的负载情况,如果负载在短时间内发生较大的变化,则启动自适应机制,不再等待周期结束,而是即刻启动负载均衡检测机制查看是否出现过载现象:如果是,则进行目录子树的迁移.实验使用 mdtest 针对 MDS1 进行 5 分钟的元数据测试,实验期间出现两次元数据请求激增,分别在启用自适应机制和未启用自适应机制的情况下观察 MDS 负载的变化,如图 7 所示.

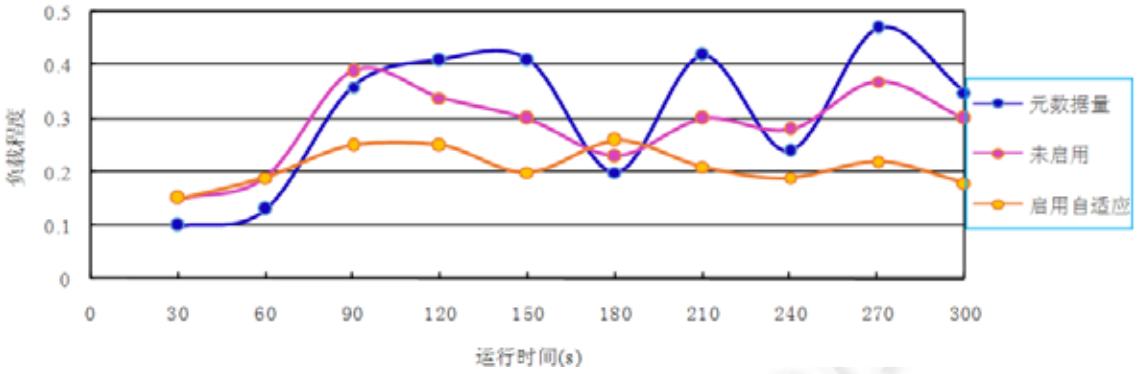

Fig.7 Experiment of adaptive mechanism

图 7 自适应机制实验

从图 7 可以看出,采用自适应机制的 MDS,在元数据请求激增后的一小段时间内就开启负载均衡机制,迁移了部分目录子树,从而降低了系统的负载;而未启用自适应机制的 MDS 则反应比较慢,无法快速转移过高的元数据热度,因此所有访问该 MDS 元数据服务的客户端的请求都会受到一定的影响.实验结果表明,采用自适应机制可以快速应变 MDS 负载的短时间激增.

为了验证本文提出的负载均衡策略在减小系统性能抖动方面的优势,我们对 Ceph 的负载均衡策略和本文的负载均衡策略进行比较,分别统计两种方法的子树迁移次数,我们可以从图 8 清晰地看出两种方法在不同时间段内进行目录子树迁移的次数的不同.

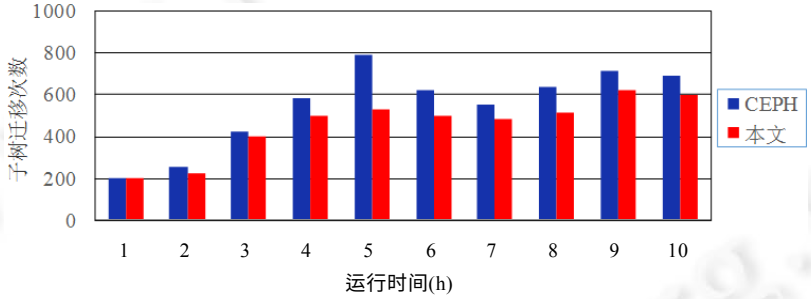

Fig.8 Migration number of directory subtrees migration

图 8 目录子树迁移次数

从图 8 可以看出,相对于 Ceph 的元数据负载均衡策略而言,本文提出的算法在同样的元数据请求下需要的子树迁移次数更少,这种方法不仅纠正了严重的系统性能抖动,而且减少了子树迁移中所消耗的 CPU、内存、I/O 资源和网络流量.

4 总 结

文件系统作为存储系统中重要的一个分支,其性能表现一直是业界、学界研究的热点.而在文件系统中,文件元数据的处理性能对整个文件系统性能有重要的影响.我们可以使用元数据服务集群去实现文件系统元数据服务性能的可扩展性,同时解决单点故障问题.与此同时,一个高效的文件系统元数据负载均衡策略使得元数据服务集群可以最大化地利用其物理资源,避免不必要的开销.本文提出了一个新的文件系统元数据负载均衡策略,其中包括一种基于服务器负载变化的检测周期自适应调整机制和一种新的自适应元数据服务负载均衡算法.最后,通过对比实验证明了该策略的可实施性、有效性和稳定性.

References:

[1] Jiang T, Hou R, Zhang LX, Zhang K, Chen LC, Chen MY, Sun N. Micro-Architectural characterization of desktop cloud workloads. In: Proc. of the 2012 IEEE Int'l Symp. on Workload Characterization (IISWC). 2012. 131−140. [doi: 10.1109/IISWC.2012.6402917]

[2] Wang F, Nelson M, Oral S, Atchley S, Weil S, Settlemyer BW, Caldwell B, Hill J. Performance and scalability evaluation of the Ceph parallel file system. In: Proc. of the 8th Parallel Data Storage Workshop. New York, 2013. 14−19. [doi: 10.1145/2538542.2538562]

[3] Ghemawat S, Gobioff H, Leung ST. The Google file system. In: Proc. of the 19th ACM Symp. on Operating Systems Principles (SOSP 2003). New York, 2003. 29−43.

[4] Shvachko K, Kuang H, Radia S, Chansler R. The hadoop distributed file system. In: Proc. of 2010 IEEE the 26th Symp. on Mass Storage Systems & Technologies. 2010(11):1−10. [doi: 10.1109/MSST.2010.5496972]

[5] Weil SA, Brandt SA, Miller EL, Long DDE, Maltzahn C. Ceph: A scalable, high-performance distributed file system. In: Proc. of the 7th Symp. on Operating Systems Design and Implementation. Washington, 2006. 307−320.

[6] Liu JL, Zhang YL, Yang L, Guo MY, Liu ZJ, Xu L. SAC: Exploiting stable set model to enhance cache files. Journal of Computer Science & Technology, 2014,29(2):293−302. [doi: 10.1007/s11390-014-1431-z]

[7] Kim T, Noh SH. pNFS for everyone: An empirical study of a low-cost, highly scalable networked storage. Int'l Journal of Computer Science & Network Security, 2014,14(3):52−59.

[8] Sun Microsystems, Inc. Lustre file system: High-Performance storage architecture and scalable cluster file system. 2008. https://www.sun.com/offers/docs/LustreFileSystem.pdf

[9] Rogers GL, Hanley J, Mohr R. Data management practices on large-scale lustre scratch file systems. In: Proc. of the 14th Conf. on Extreme Science and Engineering Discovery Environment (XSEDE 2014). Atlanta, 2014. 1−6. [doi: 10.1145/2616498.2616545]

[10] Thomson A, Abadi DJ. CalvinFS: Consistent WAN replication and scalable metadata management for distributed file systems. In: Proc. of the 13th USENIX Conf. on File and Storage Technologies (FAST 2015). Santa Clara, 2015. 1−14.

[11] Thomson A, Diamond T, Weng SC, Ren K, Shao P, Abadi DJ. Calvin: Fast distributed transactions for partitioned database systems. In: Proc. of ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2012). Scottsdale, 2012. 1−12. [doi: 10.1145/2213836.2213838]

[12] Yuan J, Zhan Y, Jannen W, Pandey P, Akshintala A, Chandnani K, Deo P, Kasheff Z, Walsh L, Bender MA, Farach-Colton M, Johnson R, Kuszmaul BC, Porter DE. Optimizing every operation in a write-optimized file system. In: Proc. of the 14th USENIX Conf. on File and Storage Technologies (FAST 2016). Santa Clara, 2016. 1−14.

[13] Jannen W, Yuan J, Zhan Y, Akshintala A, Esmet J, Jiao Y, Mittal A, Pandey P, Reddy P, Walsh L, Bender M, Farach-Colton M, Johnson R, Kuszmaul BC, Porter DE. BetrFS: A right-optimized write-optimized file system. In: Proc. of the 13th USENIX Conf. on File and Storage Technologies (FAST 2015). Santa Clara, 2015. 301−315.

[14] Esmet J, Bender MA, Farach-Colton M, Kuszmaul BC. The TokuFS streaming file system. In: Proc. of the USENIX Conf. on Hot Topics in Storage & File Systems. 2012. 1−5.

[15] Liu J, Zhang JW, Shao BQ, Dong HQ, Liu ZJ, Xu L. Metadata server clustering system for EB-scale storage. Zhong Guo Ke Xue/Science China, 2015,45(6):721−738 (in Chinese with English abstract). [doi: 10.1360/N112014-00330]

[16] Liu Z, Zhou XM. A metadata management method based on directory path. Ruan Jian Xue Bao/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]

[17] Chen T, Xiao N, Liu F. Adaptive metadata load balancing for object storage systems. Ruan Jian Xue Bao/Journal of Software, 2013, 24(2):331−342 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4177.htm [doi: 10.3724/SP.J.1001.2013.04177]

[18] Rodeh O, Teperman A. zFS-A scalable distributed file system using object disks. In: Proc. of the IEEE Nasa Goddard Conf. on Mass Storage Systems & Technologies. 2003. 207−218.

[19] Zhang X, Li L, Wang S, Yang F. Improved selective randomized load balancing in mesh networks. ETRI Journal, 2007,29(2):255−257. [doi: 10.4218/etrij.07.0206.0215]

[20] Zhang JL, Qian W, Xu XH, Wan J, Yin YY, Ren YJ. WLBS: A weight-based metadata server cluster load balancing strategy. Int'l Journal of Advancements in Computing Technology, 2012,4(1):77−85. [doi: 10.4156/ijact.vol4.issue1.9]

[21] Hua Y, Zhu Y, Jiang H, Feng D, Tian L. Supporting scalable and adaptive metadata management in ultralarge-scale file systems. IEEE Trans. on Parallel and Distributed Systems, 2011,22(4):580−593. [doi: 10.1109/TPDS.2010.116]

[22] Li B, He Y, Xu K. Distributed metadata management scheme in cloud computing. In: Proc. of the 6th Int'l Conf. on IEEE Pervasive Computing and Applications (ICPCA). 2011. 32−38. [doi: 10.1109/ICPCA.2011.6106475]

附中文参考文献:

[15] 刘健,张军伟,邵冰清,董欢庆,刘振军,许鲁.支持 EB 级存储的元数据服务器集群系统.中国科学:信息科学,2015,45(6):721−738. [doi: 10.1360/N112014-00330]

[16] 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]

[17] 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 [doi: 10.3724/SP.J.1001.2013.04177]

佘楚玉(1980－),女,广东潮州人,博士,讲师,主要研究领域为云存储,云安全.

刘育擘(1991－),男,博士生,CCF 学生会员,主要研究领域为高性能存储,高性能计算,大数据.

温武少(1969－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为云计算,计算机网络,计算系统安全,大数据.

贾殷(1991－),男,硕士,主要研究领域为CDN,SDN,分布式存储.

肖扬(1989－),男,硕士,主要研究领域为云计算,存储.