基于对象存储系统的动态负载均衡算法*)

覃灵军　冯　丹　曾令仿　刘　群

(华中科技大学信息存储系统教育部重点实验室　武汉 430074)

摘　要　负载均衡是大规模基于对象存储系统必须要考虑的重要问题。本文为此以系统总响应时间为代价函数,以对象被访问频率为依据,建立了一种将对象复制与对象迁移统一在内的动态负载均衡模型,并充分利用存储设备的智能实现系统的动态负载均衡。仿真结果表明,在存在大量热点访问和对象分布不均匀的情况下,启用对象复制和对象迁移的负载均衡算法能最大程度地减少系统的平均总响应时间。

关键词　基于对象存储,负载均衡,对象复制,对象迁移

Dynamic Load Balancing Algorithm in Object-Based Storage System

QIN Ling-Jun　FENG Dan　ZENG Ling-Fang　LIU Qun

(Huazhong University of Science and Technology, Key Laboratory of Data Storage System, Ministry of Education, Wuhan 430074)

Abstract　Load balancing is an important issue for large-scale Object-Based Storage Systems. A uniform dynamic load balancing model is proposed. Based on object access frequency, the model uses total response time as its cost function, and considers object replication and migration. The simulation results show that the algorithm using object replication and migration can dramatically minimize the mean total response time under the condition of hot-spot accesses and unbalanced object distribution.

Keywords　Object-based storage, Load balancing, Object replication, Object migration

1　引言

由于网络技术的发展及现代信息数据量的激增,人们对网络存储系统提出了更高要求。目前在网络存储领域出现两种趋势:基于对象存储[1]和智能存储[2]。基于对象存储采用对象接口(Object-Based),以对象为基本传输单位,克服了块级接口(SAN)和文件级接口(NAS)的缺陷,以此构建的存储系统在性能、可扩展性、安全性、跨平台能力等方面均比 SAN 和 NAS 优越。另一方面,随着集成电路技术的发展,存储设备上集成越来越多的处理能力,存储设备的智能化使得具有自管理、自学习能力的大规模存储系统成为可能。基于对象存储系统(Object-Based Storage System, OBSS)将对象存储与智能存储结合起来,可以轻易达到 PB 级的规模。但随着存储系统的规模扩大,系统内的存储节点出现负载不均衡的概率将增大,严重时会导致整个系统性能急剧下降。因此,I/O 的负载均衡是 OBSS 必须考虑的重要问题。

数据复制和数据迁移作为负载均衡的手段已得到广泛研究,但大都孤立地研究其中的某一种[3~7]。一般说来,数据迁移可有效平衡节点间的不均衡负载,但对于热点访问却无能为力。现实的应用中存在大量的热点访问,如网络现场直播热门赛事,在直播开始前后,存储系统将接收到成千上万次对同一媒体文件的访问。如果仅仅采取文件迁移,当热点文件从一个节点迁移到另一个节点时,势必造成另一个节点过载。因此,有必要将数据复制与迁移结合起来使用。已有少数研究将二者联合起来使用。文[8]提出了将文件复制,文件迁移和进程迁移考虑在内的负载均衡算法,但考虑的因素过多,算法过于复杂,因而不适用动态的实时环境。文[9]提出了一种实时策略,把文件复制作为一种特殊的文件迁移,但仅以响应时间最小为目标选择文件,进行迁移或复制,这种方案并不能识别热点文件,也没有对复制文件的 I/O 请求进行均衡。本文研究的动态负载均衡算法以最小化系统总响应时间为代价函数,同时考虑了对象复制和对象迁移,并有效地解决了热点访问问题。本文将算法运用于基于对象存储系统,充分利用存储设备的智能,实现自动的负载均衡。与其他算法相比,结合了对象复制与迁移的动态负载均衡算法将适用于更为广泛的负载状况(VOD 流媒体访问、随机不均匀访问等)。

2　基于对象存储系统(OBSS)

基于对象的存储系统结构如图 1 所示,由高速网络将元数据服务器(MetaData Server, MDS)、基于对象存储设备(Object-Based Storage Device, OBSD)和客户连接起来。

OBSS 实现一个基于对象的分布式文件系统,MDS 提供全局名字空间、管理文件到对象的映射,提供身份验证等安全机制;OBSD 则向外提供对象接口,以对象作为存取单位。客户是存储系统的使用者,向系统发文件请求。客户访问文件时先向元数据服务器发送请求,获取文件信息(如文件由几个对象组成以及对象所在的设备 ID 等)及访问证书,然后客户与 OBSD 直接交互。OBSD 收到客户请求后,对其身份进行认证,然后执行客户的对象读写请求。在这里,客户向 OBSD 发送的 I/O 请求与基于块的 I/O 请求不同,仅包括对象 ID、对象的偏移地址以及长度。

*)973 国家重点基础研究发展规划项目"下一代互联网信息存储的组织模式和核心技术研究" 2004CB318201、国家自然科学基金资助项目(60273074)、优秀博士学位论文专项基金。覃灵军　博士研究生;冯　丹　教授、博导;曾令仿　博士研究生;刘　群　博士研究生。

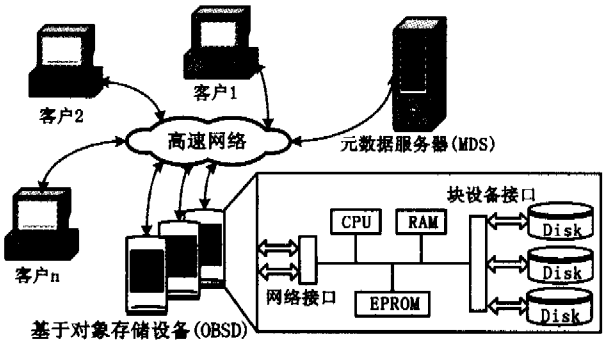

图 1　基于对象存储系统结构图

OBSD 包括 CPU、Memory、网络接口以及块设备接口，管理对象存储空间的分配与数据组织，负责将对象映射为块设备上的块。OBSD 上集成处理能力，使 OBSD 可实现智能化的负载均衡，由设备自主管理对象的复制与迁移。传统 SAN 存储系统中实现负载均衡采用集中服务器进行管理，集中服务器很容易形成瓶颈。在 OBSS 中，负载均衡是由 MDS 和 OBSD 共同完成的，MDS 完成初始对象的均匀放置，对复制对象产生均匀访问；系统运行过程中产生的负载不均衡由 OBSD 发现，并由所有的 OBSD 协同完成负载的均衡；OBSD 则在负载均衡过程结束后向 MDS 报告对象存储位置的变更。我们这里仅讨论 OBSD 上实现的负载均衡。

另外，本文讨论的负载均衡是一种实时策略，建立在低延迟的高速网络之上。随着网络技术的飞速发展（如 10G 的以太网已出现），实时的负载均衡是可行的。

3　负载均衡模型

负载的定义有多种标准，如 CPU 利用率、内存利用率等。在 OBSS 中，以 OBSD 上的对象 I/O 请求队列长度作为负载定义。当其超过一定阈值，就要进行负载均衡。OBSD 在请求队列的目标对象中挑选适合的对象进行复制，或迁移到别处，同时将涉及到该对象的 I/O 请求迁移到别处以减少负载。一般说来，对于热点对象应先进行复制，因为迁移热点对象会再度引起目的 OBSD 负载过重。当复制对象时，可能会部分选择原 OBSD 上未处理的以该对象为目标的 I/O 请求进行迁移，以最大可能均衡 I/O 请求；当迁移对象时，原 OBSD 上未处理的以该对象为目标的 I/O 请求必须全部迁移。负载均衡的模型如图 2 所示。

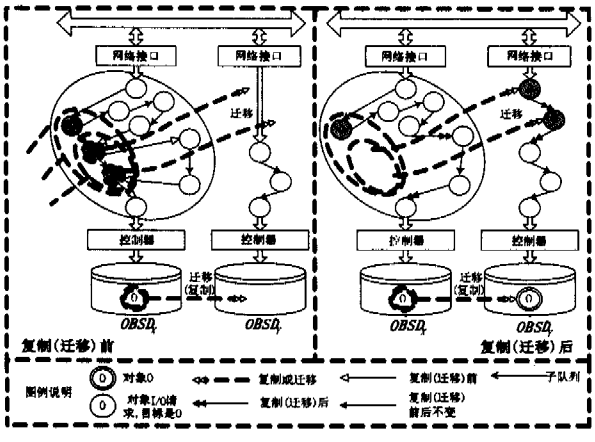

图 2　负载均衡模型

为方便描述，首先引入几个符号，如表 1 所示。

表 1　符号含义

符号	含义
$OBSD_X$	编号为 X 的 OBSD
$RQ(X)$	$OBSD_X$ 的对象 I/O 请求队列
$Rq(X,O)$	在 $RQ(X)$ 中，请求目标为对象 O 的所有 I/O 请求组成的子队列
$OS(Q)$	$\{Obj \mid \forall x \in Q, x$ 请求的目标对象是 $Obj\}$
Q_N	顺序取请求队列 Q 的前 N 个请求组成的子队列
$QL(Q)$	队列 Q 的长度
MST	对象 I/O 请求的平均处理时间，即平均服务时间（Mean Serve Time）
C_{TR}	在两个 OBSD 之间迁移一个对象 I/O 请求的平均时间
C_{TO}	在两个 OBSD 之间迁移或复制对象的平均时间

如图 2 所示，假设 $OBSD_X$ 是负载均衡的发起者，$OBSD_Y$ 是负载均衡的目标，$OBSD_X$ 将存储在其中的对象 O 复制（或迁移）到 $OBSD_Y$ 上，并把 $Rq(X, O)_{RTN}$（$1 \leqslant RTN \leqslant QL[Rq(X,O)]$），迁移至 $RQ(Y)$ 队列尾（RTN 表示需要从 $OBSD_X$ 迁移走的对象 I/O 请求总数）。下面我们以响应时间最小化为目标[9]，以对象被访问频率为依据，建立将对象复制和对象迁移统一在内的负载均衡数学模型。在我们的模型中，根据对象被访问的频率优先选择热点对象进行复制；以响应时间最小为原则，选择对象及对象 I/O 请求进行迁移。

首先定义 t 时刻 $OBSD_X$ 的响应度（Response Degree，RD）为：$RD(X,t)=\sum_{i=1}^{t时刻的QL[RQ(X)]} i \times MST$，响应度是响应时间的度量；定义 t 时刻系统的响应度（System RD，SRD）为：$SRD(t)=\sum_{OBSD_S \in 所有OBSD集合} RD(S,t)$，则系统负载均衡前后 SRD 的变化 ΔD 为：

$$\Delta D=SRD(t_{After})-SRD(t_{Before})+\Delta d=RD(X,t_{After})-RD(X,t_{Before})+RD(Y,t_{After})-RD(Y,t_{Before})+\Delta d$$

其中 Δd 为负载均衡引起的额外开销。在 OBSS 系统中，ΔD 共包含 5 部分：$Rq(X,O)_{RTN}$ 从 $OBSD_X$ 迁移至 $OBSD_Y$ 上导致 $OBSD_X$ 的响应度减少和 $OBSD_Y$ 的响应度增加；$Rq(X,O)_{RTN}$ 从 $OBSD_X$ 迁移后引起剩余队列 $RQ(X)-Rq(X,O)_{RTN}$ 的响应度减少；额外开销 Δd 即复制（或迁移）对象 O 及迁移 $Rq(X,O)_{RTN}$ 引起的开销。ΔD 可化简为下式[9]：

$$\Delta D=MST\times\{RTN^2+[QL(RQ(Y))-QL(RQ(X))+C_{TR}\times[QL(RQ(Y))+QL(RQ(X))]/MST]\times RTN\}+C_{TO}\times[QL(RQ(Y))+QL(RQ(X))] \quad (1)$$

在式(1)中，当进行对象复制时，$1 \leqslant RTN \leqslant QL[Rq(X,O)]$；当进行对象迁移时，$RTN=QL[Rq(X,O)]$。

我们的目标是使 $\Delta D<0$ 且使 ΔD 达到最小值。欲使 ΔD 取值最小，则应使下式取值最小：

$$\left|\begin{array}{l}2\times RTN-QL[RQ(X)]+QL[RQ(Y)]\\+C_{TR}\times[QL(RQ(Y))+QL(RQ(X))]/MSR\end{array}\right| \quad (2)$$

在式(2)中，C_{TR}，MST，$QL[RQ(X)]$以及 $QL[RQ(Y)]$对于 $OBSD_X$ 来说是确定的。C_{TR}，MST 为常数；$QL[RQ(X)]$是本地信息，可以从执行负载均衡的 $OBSD_X$ 获得；$QL[RQ(Y)]$是远程信息，需要通过网络获取，即 $OBSD_X$ 搜集其他 $OBSD$ 的信息，确定一个最合适目标 $OBSD_Y$，从而 $QL[RQ(Y)]$也是确定的。而 RTN 是可变的，对象复制和对象迁移的 RTN 取值是不同的，使式(2)最小的关键是确定 RTN 的值。

我们利用式(1)及(2)作为对象复制与迁移的判断准则。

对 $OBSD_X$ 进行负载均衡时，首先复制热点对象。我们以近期内(在 T_{DETECT} 时间段内)对象被访问次数(即对象被访问频率)为准则判断热点对象是否存在，如果某对象 Obj 近期内访问次数 $NA(Obj)$ 大于阈值 N_{ACCESS}，则 Obj 为热点对象。如果确定了需要复制的热点对象为 O 后，在 $1 \leqslant RTN \leqslant QL[Rq(X,O)]$ 中选择 $RTN=RTN1$，使式(2)最小，此时如有 $\Delta D<0$，则在复制对象的同时把 $Rq(X,O)_{RTN1}$ 迁移，否则仅复制对象；如果 $OBSD_X$ 不存在热点对象，则选择合适对象进行迁移，此时 $RTN=QL[Rq(X,O)]$，对于选取 $O=O1$，使得式(2)最小，将 $O1$ 及 $Rq(X,O1)$ 进行迁移。

4 OBSD 负载均衡算法的实现

首先定义两个负载阈值：QL_{ALERT} 和 QL_{MAX}，当请求队列长度 $QL \geqslant QL_{MAX}$ 时，将在 $OBSD$ 上启动负载均衡，直到 $QL < QL_{ALERT}$ 为止。

为实现负载均衡，所有的 $OBSD$ 必须实现两个进程：守护进程和负载均衡进程。守护进程完成对负载状况的实时检测；对外界 $OBSD$ 负载均衡进程发来的询问，以自身负载状况作为回应；接受或拒绝对象复制或对象迁移请求等。当守护进程检测到负载超过阈值后，启动负载均衡进程运行，本地的负载均衡进程与远程 $OBSD$ 上的守护进程协同工作，共同完成负载均衡。

假设 $OBSD_X$ 为负载均衡发起者，其负载均衡进程的具体算法如下：

$QLX=QL[RQ(X)]$；/* 本轮循环中，$OBSD_X$ 对象 I/O 请求队列长度 */

$OSX=OS[RQ(X)]$；/* 本轮循环中，在 $OBSD_X$ 上可进行复制或迁移的备选对象的集合 */

$OBSDS=\{J \mid 1 \leqslant J \leqslant N$ AND $J \neq X\}$；/* 本轮循环中所有可能的目标 OBSD 集合，其中 N 为 OBSD 总数 */

do{

1) $MOS=\Phi$；/* 本轮循环需要迁移的对象集合 */ $ROS=\Phi$；/* 本轮循环需要复制的对象集合 */ $RS=\Phi$；/* 本轮循环需要迁移的对象 I/O 请求集合 */

2) $\forall K \in OBSDS$，向 $OBSD_K$ 广播负载均衡通知；

3) $\forall K \in OBSDS$，接收 $OBSD_K$ 守护进程回应的消息，从中获取 $QL[RQ(K)]$；

4) 从 $QL[RQ(J)](1 \leqslant J \leqslant N$ AND $J \neq X)$ 中选取最小值 $QL_{MIN}=QL[RQ(Y)]$；

5) if($QL_{MIN} \geqslant QL_{ALERT}$) exit; /* 整个系统负载过重 */
else 以 $OBSD_Y$ 作为目标；

6) $HOS=\{O \mid O \in OSX$ AND $NA(O) \geqslant N_{ACCESS}\}$；

7) while($HOS \neq \Phi$){ /* 选择复制的对象 */
选择 $Obj1 \in HOS$，使得 $NA(Obj1)$ 最大；
$MOS=MOS \cup Obj1$；
对 $1 \leqslant RTN \leqslant QL[Rq(X,O)]$，取 $RTN=RTN1$，使式(2)最小；
令 $RTN=RTN1$，按式(1)计算 ΔD；
if($\Delta D<0$){
$RS=RS \cup Rq(X,O)_{RTN1}$；
$QLX=QLX-RTN1$；
if($QLX < QL_{ALERT}$)goto 9；
}
$HOS=HOS-Obj1$；
}

8) while($OSX \neq \Phi$){ /* 选择迁移的对象 */
选择 $Obj2 \in OSX$，使式(2)最小；
$ROS=ROS \cup Obj2$；$RS=RS \cup Rq(X,Obj2)$；
$QLX=QLX-QL[Rq(X,Obj2)]$；
if($QLX < QL_{ALERT}$) goto 9；
$OSX=OSX-Obj2$；
}

9) $OBSDS=OBSDS-Y$；$OSX=OSX-MOS-ROS$；

10) if($MOS \neq \Phi$) $\forall O \in MOS$，连接 $OBSD_Y$ 守护进程，复制 O 至 $OBSD_Y$；

11) if($ROS \neq \Phi$) $\forall O \in ROS$，连接 $OBSD_Y$ 守护进程，迁移 O 至 $OBSD_Y$；

12) if($RS \neq \Phi$) $\forall R \in RS$，连接 $OBSD_Y$ 守护进程，迁移 R 至 $RQ(Y)$ 队列尾；

13) 向 MDS 报告 MOS 和 ROS 中对象新位置的信息；

} while($QLX \geqslant QL_{ALERT}$)

在上述算法中，每次循环确定了目标 OBSD 后，对 I/O 请求队列进行扫描，把确定进行复制的对象 ID 加入集合 MOS；把进行迁移的对象 ID 加入集合 ROS；把进行迁移的 I/O 请求加入集合 RS。在一轮循环结束后，根据 MOS、ROS 和 RS 中的记录进行对象的复制和迁移。循环继续进行，直至 $QLX < QL_{ALERT}$。

5 仿真与结果

我们编写仿真程序，验证上述负载均衡算法，分别考查对象复制和对象迁移算法及两者的结合在不同负载状况下对平均系统响应度(Mean SRD，MSRD)的影响。

假设系统中的 OBSD 数为 $N=50$，对象总数为 1000(ID 从 0 至 999)，系统中对象 I/O 请求服从泊松到达，到达率 $\lambda=0.25$，为避免整个系统负载过载，我们让 OBSD 的服务速率约等于 λ/N，即 OBSD 的服务时间服从均值 $MST=200$ 的指数分布(用 Matlab 产生)。N_{ACCESS}/T_{DETECT} 表示对象被访问的频率，如果对象被访问的频率足够大，该对象可认为是热点对象，我们保守设定 $N_{ACCESS}/T_{DETECT}=\lambda/N$。另外，$T_{DETECT}$ 为采样期，T_{DETECT} 越小，对热点访问就越灵敏。因此，我们取 T_{DETECT} 为较小值 1000，从而得到 $N_{ACCESS}=5$。阈值 QL_{MAX} 和 QL_{ALERT} 的设定对系统响应时间有很大影响，一般根据 OBSD 的处理能力设定。在我们的仿真环境中，经验证，当 $QL_{MAX}=10$，$QL_{ALERT}=5$ 时较为合适。最后，我们这里仅关注算法本身而忽略网络延迟，故令 $C_{TR}=0$，$C_{TO}=0$。

下面讨论系统工作负载的产生。

对象初始分布状态影响系统负载的均衡性，仿真程序使用了两种分布，分别记为 $A_i(i=1,2)$。A1 表示 1000 个对象均匀分布在 50 个 OBSD 上；A2 表示对象不均匀分布，以此模拟随机不均匀访问。

为确定每个对象 I/O 请求访问的对象，仿真程序将输入一个对象 ID 序列。当 I/O 请求到达时，依次从序列中读取一个对象 ID，然后请求被发往该对象所在的 OBSD。因此，使用不同的对象 ID 序列，会产生不同程度的不均衡负载。我们用 Matlab 产生三种对象 ID(0～999)序列，输入到仿真程序中，分别记为 $Bi(i=1,2,3)$。B1 为均匀随机序列。B2 序列服从

Zipf 分布[10]。一般认为，VOD 等连续媒体节目的访问概率服从 Zipf 分布，因此 B2 中存在热点对象。B3=B1+B2，即把 B2 插入到 B1 中合成得到。

我们合成三种负载，记为 $Li(i=1,2,3)$，其中 L1=A2+B1；L2=A1+B2；L3=A2+B3。在负载 L1 下，系统的负载不均衡主要是由对象分布不均匀（或者随机不均匀访问）造成；在负载 L2 下，系统的负载不均衡主要是由大量热点访问造成；在负载 L3 下，两种情况相当。我们考查在上述三种负载状况下负载均衡算法（①无负载均衡，②仅启用对象复制，③仅启用对象迁移，④启用对象复制与迁移）的效果，即绘制从 0 至 t 时刻 MSRD 随时间变化的曲线图，其结果分别如图 3、图 4 和图 5 所示。其中 MSRD 的计算式为。

$MSRD(t)=\int_0^t SRD(u)\mathrm{d}u/t$。

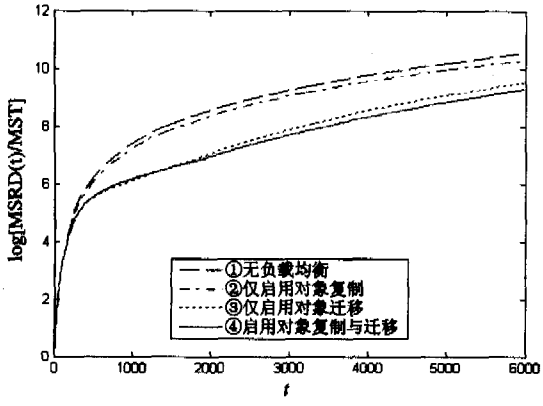

图 3　L1 负载下算法效果

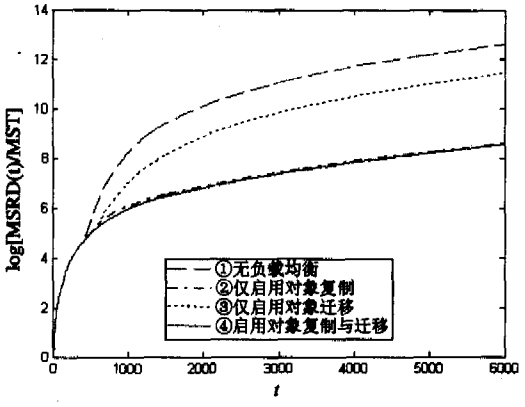

图 4　L2 负载下算法效果

从图 3、图 4 和图 5 可以看到，从 0 时刻开始的一段时间内，所有 OBSD 均未过载，不启用算法和启用算法的效果是一样的。随着 OBSD 出现负载不均衡，采用负载均衡算法的效果好于无负载均衡的情况。如图 3，在无热点访问情况下，算法②与①的曲线、算法③与④的曲线很接近。这说明对象复制不起作用，而对象迁移效果显著。在图 4 情况下，由于存在大量热点访问，对象复制对系统负载均衡起主导作用，因此②和④很接近，且效果明显好于其余两种，这说明在存在大量热点访问情况下，仅仅依靠对象迁移进行负载平衡是不够的。在图 5 情况下，热点访问与对象分布不均，共同引起系统负载失衡，对象复制与对象迁移都起到显著作用，但任何一方都不可能取代另一方。另外，图 5 中的②与③存在一个交点，这是因为从 OBSD 开始出现负载不均衡算起的一段时间内，虽然存在热点访问，但所有对象的访问次数均未达到设定的阈值 N_{ACCESS}，因此对象复制暂时不起作用，此时曲线②暂时位于③之上。但随着热点访问大量到来，对象复制开始起作用，于是曲线②上升趋势变缓，最终位于③之下。在图 5 中值得注意的是，曲线④位于所有其他曲线之下，即平均系统响应度最小，这是由于④中的对象复制与迁移共同作用的结果。例如，在 $t=6000$ 时，④的平均系统响应度比①减少了 92.33%，而②比①减少了 85.61%，③仅比①少了 60.18%。

综上所述，在存在大量热点访问和对象分布不均匀的情况下，启用对象复制和对象迁移的负载均衡算法能最大程度地减少平均系统响应度。

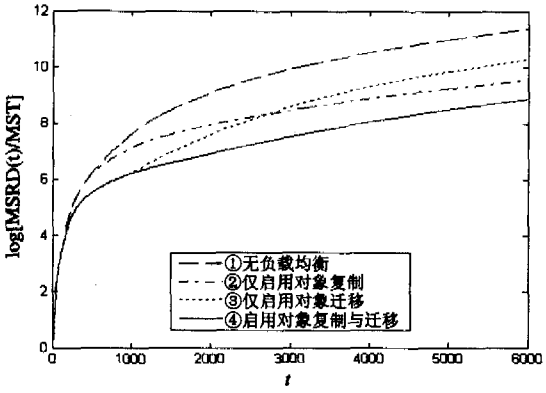

图 5　L3 负载下算法效果

结束语　我们提出一种应用于对象存储系统的动态负载均衡算法，充分利用存储设备的处理能力，达到系统的自动负载均衡。本文提出的负载均衡算法将对象复制与迁移统一在内，对由大量热点访问及对象分布不均（或随机不均匀访问）引起的系统负载失衡效果显著。

参 考 文 献

1　Mesnier M, Ganger G R, Riedel E. Object-based storage. IEEE Communications Magazine, 2003, 41(8): 84 ~ 90

2　Riedel E, Faloutsos C, Gibson G A, et al. Active disks for large-scale data processing. Computer, 2001, 34(6): 68~74

3　Loukopoulos T, Ahmad I. Static and adaptive data replication algorithms for fast information access in large distributed systems. The 20th International Conference on Distributed Computing Systems, Taipei, 2000

4　Stockinger H, Samar A, Allcock B, et al. File and object replication in data grids. 10th IEEE International Symposium on High Performance Distributed Computing, San Francisco, California, 2001

5　Lamehamedi H, Szymanski B, Shentu Z, et al. Data replication strategies in grid environments. Fifth International Conference on Algorithms and Architectures for Parallel Processing, Beijing, China, 2002

6　Kalogeraki V, Melliar-Smith P M, Moser L E. Dynamic migration algorithms for distributed object systems. 21st International Conference on Distributed Computing Systems, Phoenix, Arizona, USA, 2001

7　Griffioen J, Anderson T A, Breitbart Y A. A dynamic migration algorithm for a distributed memory-based file management system. Seventh Inernational Workshop on Research Issues in Data Engineering, Birmingham, UK, 1997

8　Hac A. A distributed algorithm for performance improvement through file replication, file migration, and process migration. IEEE Transactions on Software Engineering, 1989, 15(11): 1459 ~1470

9　Hurley R T, Soon Aun Yeap. File migration and file replication: a symbiotic relationship. IEEE Transactions on Parallel and Distributed Systems, 1996, 7(6): 578 ~ 586

10　Rabinovich M, Rabinovich I, Rajaraman R, et al. A dynamic object replication and migration protocol for an Internet hosting service. 19th IEEE International Conference on Distributed Computing Systems, Austin, Texas, USA, 1999