面向云应用的存储缓存子系统

蔡 涛 牛德姣 张永春 倪晓蓉 周东明

(江苏大学计算机科学与通信工程学院 镇江 212013)

摘 要 性能是云应用的重要指标,云应用与现有应用在数据存储与管理方式和访问模式等方面存在较大差异,这使得传统的缓存管理算法难以适应云应用的要求。针对云应用的特性,设计了基于影响因子的缓存策略,亦即将元数据和数据缓存分开管理,使用影响因子综合管理影响缓存被再次访问的几率的多种因素,区别创建、打开、读取和修改等操作对缓存再次访问的几率的影响;设计了缓存关联管理策略,即利用元数据和数据之间的关联提高缓存管理的性能;设计了缓存主动调度策略,即通过主动淘汰较低影响因子的缓存项和动态调整元数据与数据缓存的大小来提高缓存子系统的适应能力和性能。最后实现了原型系统,并使用 Filebench 和 Postmark 进行了测试和分析,验证了面向云应用缓存子系统原型能提高 1%~120% 的 I/O 性能以及 2%~87% 的操作处理速度。

关键词 缓存,云应用,文件系统

中图法分类号 TP316.4 **文献标识码** A

Storage Caching Sub-system for Cloud Application

CAI Tao NIU De-jiao ZHANG Yong-chun NI Xiao-rong ZHOU Dong-ming

(School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China)

Abstract Performance is very important for the cloud application. There are some differences between cloud and current application in management strategy and access pattern, which leads to the traditional cache management strategy can't adapt to the requirement of the cloud application. According to the features of cloud application, the cache strategy based on impact factors was designed to implement the storage cache sub-system for cloud application. Metadata and data cache were used to manage separately, making a distinction about the impact of the creation, opening, reading, writing and modifying on the probability of the cache accessed again. Then the two cache correlation strategics were designed to improve the performance of cache management according to the relation between metadata and data. The actively scheduling strategy was used to improve the adaptability based on eliminating the low impact factor cache entries actively and adjusting the space of metadata cache and data cache dynamically. At last, the prototype was released and was evaluated by Filenench and Postmark. The result shows the storage cache sub-system can improve 1%~120% I/O performance and 2%~87% operation processing speed.

Keywords Cache, Cloud application, File system

移动设备和网络改变了人们获得计算和存储资源的方式。但移动设备的计算和存储能力有限,需要各类云应用作为支撑;大量移动设备产生的海量的数据,需要借助各类云应用进行存储、管理、分享和推广;因此高性能是云应用的重要要求之一。海量存储系统负责存储和管理云应用的海量数据,是各类云应用的支撑之一,也是影响云应用性能的重要因素。

缓存是提高海量存储系统 I/O 性能的重要方法。缓存的目标是尽可能保存用户后续访问中最需要的数据,因此用户访问数据的方式和习惯是决定缓存管理方法的重要因素之一。云应用与现有应用有很大的区别,云系统中数据量的增加非常迅速,但单个数据被访问的频率却急剧下降,此外在读写次数的比例、元数据和数据被访问的频率、用户访问数据的习惯等方面都发生了很大的改变,这使得现有缓存管理方法难以适应云应用的要求。

我们首先分析云应用中数据存储、管理和用户访问特性,给出相关定义,设计基于影响因子的缓存策略、关联淘汰策略和主动调度算法,并在面向对象存储设备的基础上添加缓存子系统原型,使用通用测试工具和数据集进行测试与分析。

1 相关工作

在分布式文件系统的缓存研究方面。文献[1]针对分布式文件系统中的二级缓存提出了 MQ 策略。文献[2]提出了 NFS-CD,其在 NFSv4 的基础上实现了支持快速提交和容错

到稿日期:2013-05-29 返修日期:2013-08-18 本文受高等学校博士学科点专项科研基金(20093227110005),广东省自然科学基金(S2011010006118),江苏省高校自然科学基金(09KJB520001),浙江省自然科学基金(LY13F020012)资助。

蔡 涛(1976-),男,博士,副教授,CCF 会员,主要研究方向为海量存储系统、大数据;牛德姣(1978-),女,博士生,讲师,主要研究方向为海量存储系;张永春(1988-),男,硕士生,主要研究方向为海量存储系统;倪晓蓉(1989-),女,硕士生,主要研究方向为海量存储系统。

的分布式协作缓存。文献[3]提出了 RACE，其将全局 I/O 请求特性和单个文件 I/O 请求特性相结合提高了缓存的命中率。文献[4]在 Blue Gene 系统中增加了客户端和 I/O 结点端缓存实现了应用与数据的分段。文献[5]针对集群环境，提出了两阶段缓存管理策略，分别实现了 I/O 的质量管理和缓存分配。文献[6]提出了客户端缓存协作策略，从而提高了分布式文件系统的性能。文献[7]设计了 Panache，实现了分布式文件系统中的并行缓存。

在网格文件系统的缓存研究方面。文献[8]使用预取机制，针对网格应用的特点设计了基于时间多项式的复制算法和分布式缓存策略，提高了对不同访问模式的适应能力。文献[9]综合预取和缓存数据，提出了 ASP 自适应条带化预取策略，提高了预取的准确性。文献[10]提出了位置敏感的协作缓存策略，提高了网络文件系统中分布式缓存的管理性能。文献[11]针对 P2P 文件系统提出了两层缓存策略，并分别设置静态区和动态区来分别缓存被很多用户访问和被高频重复访问的数据。文献[12]提出了协作缓存策略 C-LRU 和 RobinHood，实现了基于簇的缓存管理。

此外文献[13]分析了文件系统的预取对缓存策略性能的影响。文献[14]将文件系统内核态的页缓存输出到用户态，提高了系统的性能的灵活性。文献[15]提出了缓存芯片的管理策略，减少了数据服务器的能耗。文献[16]提出了基于对象的缓存策略。文献[17]针对多处理器系统提出了分层缓存结构，其减少访问数据的延迟。文献[18]提出了 NAFS，其使用新型缓存策略提高了访问固态存储系统的性能。文献[19]设计了 RAF，其使用 SSD 作为磁盘随机访问的缓存，并将读写缓存分开，减少了对 SSD 寿命的影响。

现有缓存研究主要集中在分布式文件系统、网格文件系统、预取对缓存的影响、减少能耗和与 SSD 相结合等方面，但缺乏针对云应用的缓存策略研究。而云应用与现有应用有着较大的区别，使用现有的缓存管理策略难以适应云应用的特性。

2 存储缓存子系统的分析及相关定义

云应用改变了数据存储和管理的特点，以及用户访问数据的模式。我们在分析云应用特点的基础上，给出面向云应用存储缓存策略的相关定义。

2.1 云应用的特性

首先，在云应用的数据操作类型方面，相对现有应用中 4∶1的读写比例，云应用中的读写比例是 2∶1，同时元数据操作数量占到了访问操作总量的 2/3 以上，因此云应用中写操作和元数据操作更频繁；其次，在访问的数据大小方面，云应用中超过 75%的用户访问的文件小于 20kB，超过 30%的操作读写的数据量小于 20kB，而其他应用中操作读写的数据量则接近于 10MB，因此云应用中对随机读写的性能要求更高；此外在数据的生存期方面，云应用中被删除的文件绝大部分生存期小于 1 天，50%被删除的文件生存期小于 100ms，因此最终被删除文件的生存期较短；最后，在被访问次数方面，云应用中 90%以上的数据在产生之后不会被再次访问，超过 50%文件被多次打开的时间间隔小于 200ms，93%以上的文件只会被一个或两个用户访问，因此文件被同时访问的几率很低，这使得严格的锁协议变得不再非常重要。总的来说，云应用中写操作和元数据操作的比例更高，对随机读写性能的要求更高，绝大部分文件不会被多次打开，但文件被多次打开的时间间隔较小，被删除文件的生存期较短；这些特点与其他应用有着较大的差别，使用现有的缓存管理算法难以适应，需要针对云应用的特点，设计和实现新型的存储缓存子系统。

我们引入缓存项影响因子，作为管理云应用存储缓存的依据；针对元数据操作多于数据访问操作的特点，将元数据和数据缓存分开，设计相应的管理策略；针对写操作更频繁的特性，区分读写操作改变的缓存项影响因子值；针对数据生存期的变化，区别创建和再次打开操作所改变的缓存项影响因子值；针对单个文件被访问次数的减少，主动淘汰较长时间未再次访问的数据；此外利用云应用中小文件比例高、被删除文件生存期较短等特性，利用存储缓存提高文件系统的性能。

2.2 相关定义

在分析云应用特性的基础上，我们给出存储缓存子系统中相关概念的定义如下：

定义 1 元数据缓存项影响因子 pm（$pm\in N$ 且 $pm>1$）。

定义 2 数据缓存项影响因子 pd（$pd\in N$ 且 $pd>1$）。

定义 3 元数据缓存 MC（$MC=\{(m_i,pm_i,m\_updata)\mid i=1,\cdots,n,n\in N\}$），由 n 个（m_i，pm_i，m_updata）组成，m_i 表示一个缓存的元数据项，pm_i 是元数据缓存项的影响因子，$m_updata\in\{0,1\}$表示该元数据缓存项是否被修改过的标志（0 表示未修改，1 表示修改过）。

定义 4 数据缓存 DC（$DC=\{(d_i,pd_i,d\_updata)\mid i=1,\cdots,l,l\in N\}$），由 1 个（$d_i$，$pd_i$，$d\_updata$）组成，$d_i$ 表示一个数据缓存项，pd_i 是数据缓存项的影响因子，$d_updata\in\{0,1\}$表示该数据缓存项是否被修改过的标志（0 表示未修改，1 表示修改过）。

定义 5 元数据缓存项影响因子的变化单位 pm_T（$pm_T\in N$ 且 $pm_T>1$）。

定义 6 数据缓存项影响因子的变化单位 pd_T（$pd_T\in N$ 且 $pd_T>1$）。

定义 7 数据缓存管理周期 T（$T\in N$ 且 $T>1$）。

定义 8 元数据缓存管理周期 T_m（$T_m\in N$）。

定义 9 缓存调整阈值 C（$C\in N$），作为是否调整元数据缓存和数据缓存的标准。

定义 10 元数据缓存主动淘汰阈值 E_m（$E_m\in N$），作为主动淘汰元数据缓存项的标准。

定义 11 数据缓存主动淘汰阈值 E_d（$E_d\in N$），作为主动淘汰数据缓存项的标准。

定义 12 元数据缓存写回阈值 WB_m（$WB_m\in N$），作为主动写回元数据缓存项的标准。

定义 13 数据缓存写回阈值 WB_d（$WB_d\in N$），作为主动写回数据缓存项的标准。

定义 14 参数 β，满足条件 $\beta\in N$ 且 $\beta>1$，是读取缓存时增加的缓存影响因子值相对影响因子标准值的比例。

定义 15 参数 ε，满足条件 $\varepsilon\in N$ 且 $\varepsilon>\beta$，是修改缓存时所增加的缓存影响因子值相对影响因子标准值的比例。

3 基于影响因子的缓存策略

云应用中元数据操作数量比数据操作多一倍，元数据的

访问更频繁，这是由于用户更多地只是访问和搜索文件名、创建和修改时间等元数据，因此云应用中元数据和数据访问特性存在差异。我们将元数据和数据缓存分开，设计相应的管理策略。此外微博等云应用中用户更多使用追加方式发布信息，这使得写操作的比例更高；同时大量文件在产生后不被再次访问，但再次打开文件的时间间隔更小，被删除文件的生存期也很短，因此不同的数据访问方式使得数据再次被访问的几率也不相同。依据单一因素管理缓存难以适应云应用的要求，我们使用影响因子综合和有区别地管理各种因素对缓存项被再次访问几率的影响，针对元数据和数据缓存的不同特性，分别设计相应的影响因子管理策略，用于管理元数据和数据缓存。

3.1 元数据缓存影响因子管理策略

使用式(1)计算元数据缓存项影响因子的更新周期 T_m 值。

$$T_m = T/\delta \tag{1}$$

式中，参数 δ，满足条件 $\delta \in N$ 且 $\delta > 1$，是数据与元数据缓存更新周期的比例。

在每个 T_m，使用式(2)更新元数据缓存项影响因子 pm 的值。

$$pm = pm/\alpha \tag{2}$$

式中，参数 α，满足条件 $\alpha \in N$ 且 $\alpha > 1$，是元数据缓存项影响因子衰减的速率。

建立新文件时，将对应的元数据存入元数据缓存 MC，同时设置该元数据缓存项的缓存影响因子 pm 的值为 pm_T，并将 MC 中对应项的 m_updata 值设置为 1。

读取元数据时，检查是否已缓存了该元数据，如没有，则从存储设备中读取元数据并存入元数据缓存中，并设置影响因子 pm 的值为 pm_T/β；否则从缓存直接返回元数据，并使用式(3)更新元影响因子 pm 的值。

$$pm = pm + pm_T/\beta \tag{3}$$

修改元数据时，检查是否已缓存了该元数据，如没有则将新元数据值存入元数据缓存中，并设置影响因子 pm 的值为 pm_T/ε；如已存在，则修改相应的缓存，将 m_updata 的值设置为 1，并使用式(4)更新缓存影响因子 pm 的值。

$$pm = pm + pm_T/\varepsilon \tag{4}$$

删除元数据时，检查是否已缓存了该元数据，如存在，则先删除对应的元数据缓存项、缓存影响因子 pm 和修改标志 m_updata，并删除存储设备中存储的元数据。

3.2 数据缓存影响因子管理策略

在每个缓存检查周期 T，使用式(5)更新所有数据缓存项影响因子 pd 的值。

$$pd = pd/\varphi \tag{5}$$

式中，参数 φ，满足条件 $\varphi \in N$ 且 $\varphi > 1$，是数据缓存项影响因子衰减的速率。

创建文件后，将对应的数据存入数据缓存 DC，同时设置该数据缓存项的缓存影响因子 pd 的值为 pd_T，并将 DC 中对应项的 d_updata 值设置为 1。

读取数据时，检查是否已缓存了相应的数据，如没有，则从存储设备读取数据并存入数据缓存中，设置影响因子 pd 的值为 pd_T/β；否则从缓存直接返回元数据，并使用式(6)更新数据缓存项缓存影响因子 pd 的值。

$$pd = pd + pd_T/\beta \tag{6}$$

修改数据时，检查是否已缓存了该数据，如没有，则将数据写入数据缓存中，并设置影响因子 pd 的值为 pd_T/ε；如已存在，则修改相应的数据缓存，设置 d_updata 的值为 1，并使用式(7)更新数据缓存项影响因子 pd 的值。

$$pd = pd + pd_T/\varepsilon \tag{7}$$

删除数据时，检查是否已缓存了该数据，如存在，则先删除对应的数据缓存项、缓存影响因子 pd 和修改标志 d_updata，并删除存储设备中存储的数据。

4 缓存关联管理策略

云应用中元数据与数据的访问特性不同，但彼此又相互关联。元数据操作一般伴随着数据操作出现，但元数据操作不一定会导致相应的数据操作。因此面向云应用的存储缓存子系统中，在分开管理元数据和数据缓存之后，还需要设计元数据和数据缓存的关联管理策略，以提高缓存子系统的性能。

删除元数据后相应数据被删除的概率很大，可以主动淘汰相应的数据缓存；读取数据后访问相应元数据的概率很高，应该将对应的元数据主动调入缓存；修改数据时一般都会伴随修改相应元数据，因此主动将元数据调入缓存能提高缓存子系统的性能。我们设计缓存关联管理策略来协调管理元数据与数据缓存，算法具体流程如下。

```
Cache_Relation_Management(operation_type)
{
  If (operation_type 是删除元数据)
  {
    While (存在对应的数据缓存)
    {
      If (数据缓存项的值 d_updata 为 1)
        将数据缓存项的内容写回到磁盘;
      删除数据缓存项以及对应的 pd 和 d_updata ;
    }
  }
  If (operation_type 是读数据 and 未缓存对应的元数据)
  {
    在元数据缓存中调入对应的元数据;
    设置 pm 的值为 pm_T/β;
  }
  If (operation_type 是修改数据 and 未缓存对应的元数据)
  {
    在元数据缓存中调入对应的元数据;
    设置 pm 的值为 pm_T/ε;
  }
}
```

使用缓存关联管理策略，能利用元数据与数据之间的关联，主动协调元数据与数据缓存，提高存储缓存子系统的性能。

5 缓存主动调度策略

存储缓存子系统中包括元数据与数据缓存，简单设置静态的元数据与数据缓存大小，难以适应云应用的动态变化。缓存中影响因子较低的缓存项在缓存空间不足时才进行淘汰，占用了大量的缓存空间，影响了访问缓存中数据与元数据的性能，也增加了调入数据和元数据时的开销。此外，缓存中

部分修改了很久的数据与元数据在调出缓存时才被更新到磁盘，这增加了调出缓存的时间开销，也增加了掉电带来的数据丢失风险。

我们设计缓存主动调度策略，定期检查缓存的命中和访问情况，主动调整元数据和数据缓存的空间分配以及缓存项。使用 $fm(fm\in N)$ 和 $fd(fd\in N)$ 分别表示未命中元数据缓存和数据缓存操作的比例，在每个数据缓存检查周期 T，使用如下策略主动调度元数据和数据缓存。

```
Cache_scheduling(fm,fd)
{
  If (fm−fd>C)
  {
    If (存在 pm 值最小且 m_updata 值为 0 的元数据缓存)
    {
      查找 pm 值最小且 m_updata 值为 0 的元数据缓存;
      删除对应的元数据缓存项;
    }
    Else
    {
      查找 pm 值最小的元数据缓存;
      将选择出的元数据缓存项写回到磁盘;
    }
    减少数据缓存 DC 的大小;
    增加元数据缓存 MC 的大小;
  }
  If ((fd−fm)>C)
  {
    If (存在 pd 值最小且 d_updata 值为 0 的数据缓存)
    {
      查找 pd 值最小且 d_updata 值为 0 的数据缓存;
      删除对应的数据缓存项;
    }
    Else
    {
      查找 pd 值最小的数据缓存;
      将选择出的数据缓存项写回到磁盘;
    }
    减少元数据缓存 MC 的大小;
    增加数据缓存 DC 的大小;
  }
  While (存在 pm<Em 且 m_updata 值为 0 or pd<Ed，且 d_updata
  值为 0 的项)
      删除对应的元数据和数据缓存;
  If (当前系统负载较轻)
      While (pd<WBd，且 d updata 值为 1 的数据缓存)
        将数据缓存更新到磁盘;
        将数据缓存的 m_updata 和 d_updata 值设置为 0
      While (存在 pm<WBm 且 m_updata 值为 1 的元数据缓存)
        将元数据缓存更新到磁盘;
        将元数据缓存的 m_updata 和 d_updata 值设置为 0;
}
```

当元数据相比数据缓存的命中率大于设定值时，通过主动释放部分影响因子较低的元数据缓存项来减少元数据缓存，同时增加数据缓存的大小；当元数据相比数据缓存的命中率小于设定值时，通过主动释放部分影响因子较低的数据缓存项来减少数据缓存，同时增加元数据缓存的大小；从而实现对元数据与数据缓存大小的动态调节。通过删除影响因子低且未修改的元数据和数据缓存项，以及在负载较轻写回部分影响因子较低的修改后的缓存项，可以主动腾出缓存空间，提高缓存访问和调入的效率。

6 算法分析与原型测试

我们针对云应用的特点，设计了基于影响因子的缓存策略、缓存关联管理策略和缓存主动调度策略，首先给出面向云应用存储缓存子系统的结构，再从理论上分析算法的性能，同时实现原型系统使用通用数据集和测试工具进行测试与分析。

6.1 面向云应用存储缓存子系统的结构

我们在云应用的存储系统中增加缓存子系统，使用基于影响因子缓存策略、缓存关联管理策略和缓存主动调度策略，构建新型存储缓存子系统，结构如图 1 所示。

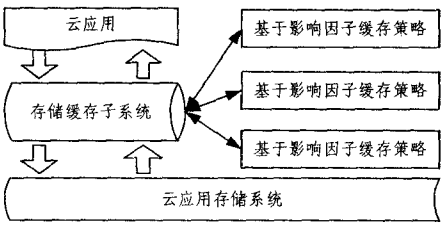

图 1 云应用存储缓存子系统结构图

存储缓存子系统截获云应用的访问请求，查找是否已缓存了相应的数据或元数据，并将相关数据反馈给云应用；同时存储缓存子系统向云应用存储系统提交淘汰的数据和元数据，并读取所需的数据或元数据；基于影响因子缓存策略、缓存关联管理策略和缓存主动调度策略用于管理存储缓存子系统，提高存储缓存子系统的效率，从而提高云应用存储系统的性能。

6.2 算法的讨论

基于影响因子的缓存策略中，将元数据和数据缓存分开管理，分别使用不同的缓存管理策略，能适应元数据和数据访问特性的不同，提高缓存的利用率；使用影响因子作为调度缓存的依据，能区分创建、打开、读取和修改等操作对缓存项再次被访问的几率的影响，综合各类影响缓存项被再次访问的因素，适应云应用的特点，解决缓存管理因素单一的问题。

缓存关联管理策略中，在删除元数据缓存项数据缓存中对应的缓存项、缓存数据，缓存对应的元数据以及修改数据的同时检查是否缓存了对应的元数据等方式，能利用云应用中元数据和数据之间的关联关系，提高缓存子系统的性能。

缓存主动调度策略中，通过主动淘汰影响因子较低的缓存项，能减少缓存调度和访问的时间开销；根据元数据和数据缓存命中率的差异，动态调整元数据和数据缓存大小的比例，能适应云应用运行时的动态变化，提高缓存子系统的适应能力和性能。

6.3 原型系统与测试

我们在开源基于对象存储设备 Open-osd 的基础上，修改 OSD Target 模块中的代码，增加元数据和数据缓存，实现面向云应用存储缓存子系统的原型。为了减少网络通信对 I/O 性能的影响，将原型系统的各组件安装在同一台计算机中进

行测试。构建测试环境时，使用四核 Intel Xeon E5606 2.13 GHz 处理器，配置 14GB 内存，安装 Redhat6.1 企业版操作系统，文件系统为 ext3。使用 Postmark 和 FileBench 作为面向云应用存储缓存子系统原型的测试工具，分别测试读写性能和操作处理速度。

首先使用 FileBench 测试缓存大小为 5M、10M 和 100M 时的 I/O 性能，应用 Fileserver 和 Varmail 两种类型的负载，设置文件大小为 1kB，线程数为 50，测试结果如图 2 和 3 所示。

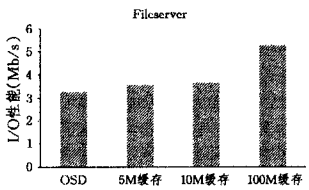

图 2 应用 Fileserver 负载时的 I/O 性能

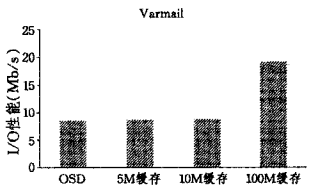

图 3 应用 Varmail 负载时的 I/O 性能

从图 2 和图 3 可知，当缓存大小为 5M、10M 和 100M 时，存储缓存子系统均能有效地提高 OSD 的 I/O 性能，提高幅度在 1%～120%之间。当缓存容量仅为 5M、应用 Fileserver 负载时，I/O 性能提高了 9%，应用小文件操作比例更高的 Varmail 负载时 I/O 性能仍然能提高 1%；当增大缓存容量后 I/O 性能的提高更明显，应用 Fileserver 负载时 I/O 性能提高了 60%，应用 Varmail 负载时 I/O 性能提高了 120%。总的说来，面向云应用的存储缓存子系统原型能有效地提高 I/O 性能，缓存容量的增加能更好地提高小文件的访问性能。

再使用 FileBench 测试缓存容量为 5M、10M 和 100M 时的操作处理速度，同样应用 Fileserver 和 Varmail 两种类型的负载，设置文件大小是 1kB，线程数为 50，测试结果如图 4 和图 5 所示。

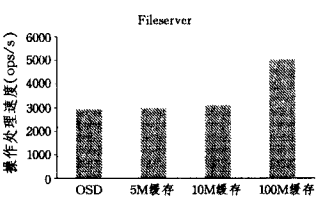

图 4 应用 Fileserver 负载时的操作处理速度

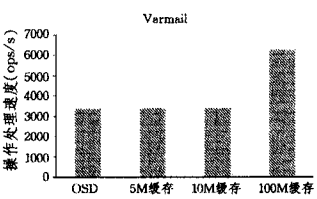

图 5 应用 Varmail 负载时的操作处理速度

从图 4 和图 5 可知，面向云应用缓存子系统能提高 2%～87%的操作处理速度。在缓存容量较小仅为 5M 时仍然能提高 OSD 的操作处理速度，应用 Fileserver 负载时提高了 2%的操作处理速度，应用小文件操作比例更高的 Varmail 负载时操作处理速度仍然提高了 2%；增加缓存容量后，操作速度的提高更加明显，应用 Fileserver 负载时操作处理速度提高了 72%，应用 Varmail 负载时操作速度提高更明显，达到了 87%。总的说来，不管缓存的大小如何，操作处理速度均得到了提高，增大缓存容量对小文件操作比例更高的应用效果更明显。

最后使用 Postmark 测试缓存容量为 5M、10M 和 100M 时的读写性能，读或写 10000 个 500B～9.77kB 大小的文件时测试结果如图 6 和图 7 所示。

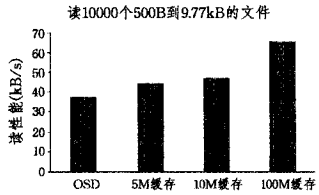

图 6 使用 Postmark 测试的读性能

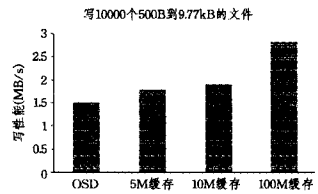

图 7 使用 Postmark 测试的写性能

从图 6 和图 7 可知，读写性能提高了 17%～86%。缓存容量较小时读性能提高了 18%，增加缓存容量后读性能提高了 74%；写性能在缓存容量较小时提高了 17%，增加缓存容量后提高了 86%。总体来说，使用面向云应用缓存子系统能提高读写性能，其中写性能的提高更明显，尤其是采用较大的缓存容量时写性能提高较多。

结束语 本文针对云应用的特性，设计了基于影响因子缓存策略、缓存关联管理策略和缓存主动调度策略，将元数据和数据缓存分开管理，使用影响因子综合影响缓存被再次访问的几率的多种因素，区别创建、打开、读取和修改等操作对缓存被再次访问的几率的影响，关联管理元数据和数据缓存，动态调整元数据和数据缓存的大小，以及通过主动淘汰较低影响因子的缓存项等方法，提高缓存子系统的性能。最后实现了原型系统，使用 Filebench 和 Postmark 测试和分析了算法的性能，验证了面向云应用缓存子系统原型能提高 1%～120%的 I/O 性能，以及 2%～87%的操作处理速度。

目前的原型系统中仅实现了存储设备端的缓存，下一步拟在云应用的服务器端也增加缓存，并研究各缓存子系统间的协作机制，进一步提高缓存系统的性能。

参 考 文 献

[1] Zhou Yuan, Chen Z, Li K. Second-level buffer cache management [J]. Parallel and Distributed Systems, IEEE Transactions, 2004, 15(6): 505-519

[2] Batsakis A, Burns R. NFS-CD: Write-Enabled cooperative caching in NFS[J]. Parallel and Distributed Systems, IEEE Transactions, 2008, 19(3): 323-333

[3] Zhu Yi-feng, Hong Jiang. Race: a robust adaptive caching strategy for buffer cache[J]. Computers, IEEE Transactions, 2008, 57(1): 25-40

[4] Saila F, Garcia Blas J, Carretro J, et al. Design and evaluation of Multiple-Level data staging for blue gene systems[J]. Parallel and Distributed Systems, IEEE Transactions, 2011, 22(6): 946-959

[5] Prabhakar R, Srikantaiah S, Garg R, et al. Adaptive QOS decomposition and control for storage cache management in Multi-server environments[C]// Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium. Newport Beach, CA, 2011: 402-413

[6] Liao Wei-keng, Ching A, Coloma K, et al. An implementation and evaluation of Client-Side file caching for MPI-IO[C]// Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International. Long Beach, CA, 2007: 1-10

[7] Eshel M, Haskin R L, Hildebrand D, et al. Panache: a parallel file system cache for global file access[C]// The 8th USENIX Conference on Files and Storage Technologies. 2010: 55-168

(下转第 210 页)

graphs[C]//COCOA 2011, LNCS 6831. Springer, 2011: 491-499

[17] Richter S, Helmert M, Gretton C. A stochastic local search approach to vertex cover[C]// Proceedings of the 30th German Conference on Artificial Intelligence (KI). 2007

[18] Guturu P, Dantu R. An impatient evolutionary algorithm with probabilistic tabu search for unified solution of some NP hard problems in graph and set theory via clique finding[J]. IEEE Transactions on Systems, Man and Cybernetics, 2008, 28: 645-666

[19] Pullan W. Approximating the maximum vertex/edge weighted clique using local search[J]. Journal of Heuristics, 2008, 14: 117-134

[20] Alphonse E, Osmani A. A model to study phase transition and plateaus in relational learning [C]// 18th International Conference on Inductive Logic Programming. 2008: 6-23

[21] Heras F, Baneres D. The impact of Max-SAT resolution-based preprocessors on local search solvers. Journal on Satisability[J]. Boolean Modeling and Computation (JSAT), 2010, 7(2/3): 89-126

[22] Zhao C, Zheng Z. Threshold behaviors of a random constraint satisfaction problem with exact phase transitions[J]. Information Processing Letters, 2011, 111(20): 985-988

[23] Cai S, Su K, Sattar A. Local search with edge weighting and configuration checking heuristics for minimum vertex cover[J]. Artificial Intelligence, 2011, 175(9/10): 1672-1696

[24] Cai S, Su K, Luo C, et al. NuMVC: an efficient local search algorithm for minimum vertex cover[J]. Journal of Artificial Intelligence Research, 2013, 46: 687-716

[25] Biere A, Heule M, Maaren H, et al. Handbook of satisfiability [M]. Amsterdam: IOS press, 2009

[26] Papadimitriou C H. On selecting a satisfying truth assignment [C]// Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science. 1991: 163-169

[27] Selman B, Kautz H. Local search strategies for satisfiability testing[C]// AMS-DIMACS Series on Discr Math Theor Comp Sci. 1996, 26: 521-531

[28] Iwama K, Tamaki S. Improved bounds for 3-SAT [C]// Proceedings of the Fifteenth AnnualACM-SIAM Symposium on Discrete Algorithms. 2004: 321-322

[29] Alekhnovich M, Ben-Sasson E. Linear upper bounds for random walk on small density random 3-cnfs[J]. SIAM J Comput, 2006, 36(5): 1248-1263

[30] Coja-Oghlan A, Feige U, Frieze A, et al. On smoothed k-CNF formulas and the walksat algorithm[C]// Proc 20th SODA. 2009: 451-460

[31] Coja-Oghlan A, Frieze A. Analyzing walksat on random formulas[C]// Proc 9th ANALCO. 2012: 48-55

[32] Kamath A, Motwani R, Palem K, et al. Tail bounds for occupancy and the satisfiability threshold conjecture[J]. Random Struct Alg, 1995, 7: 59-80

(上接第 199 页)

[8] Nukarapu D T, Bin Tang, Wang Li-qiang, et al. Data replication in data intensive scientific applications with performance guarantee[J]. Parallel and Distributed Systems, IEEE Transactions, 2011, 22(8): 1299-1306

[9] Sung Hoon Baek K P, Sung Hoon B, Kyu Ho park. Prefetching with adaptive cache culling for striped disk[C]// 2008 USENIX Annual Technical Conference. Boston, MA, USA, 2008: 363-376

[10] Jiang S, Zhang Xue-chen, Shuang Liang, et al. Improving networked file system performance using a Locality-Aware cooperative cache protocol[J]. Computers, IEEE Transactions, 2010, 59(11): 1508-1519

[11] Wei Qi-ying, Qin Ting-ting, Fujita S. A Two-Level caching protocol for hierarchical Peer-to-Peer file sharing systems[C]// Parallel and Distributed Processing with Applications (ISPA), 2011 IEEE 9th International Symposium. Busan, 2011: 195-200

[12] Anderson E, Hoover C, Li Xiao-zhou. New algorithms for file system cooperative caching[C]// Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium. Miami Beach, FL, 2010: 437-440

[13] Butt A R, Gniady C, Hu Y C. The performance impact of kernel prefetching on buffer cache replacement algorithms[J]. Computers, IEEE Transactions, 2007, 56(7): 889-908

[14] Spillane R P, Dixit S, Archak S, et al. Exporting kernel page caching for efficient user-level I/O[C]// Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium. Incline Village, NV, 2010: 1-13

[15] Yue Jian-hui, Zhu Yi-feng, Zhao Cai, et al. Energy efficient buffer cache replacement for data servers[C]// Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference. Dalian, Liaoning, 2011: 329-338

[16] Schoeberl M. A Time-Predictable object cache [C]// Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2011 14th IEEE International Symposium. 2011: 99-105

[17] Li Chong-min, Wang Hai-xia, Xue Yi-bo, et al. Fast hierarchical cache directory: a scalable cache organization for Large-Scale CMP[C]// Networking, Architecture and Storage (NAS), 2010 IEEE Fifth International Conference. 2010: 367-376

[18] Park S O, Kim S J. An efficient array file system for multiple Small-Capacity NAND flash memories[C]// Network-Based Information Systems (NBiS), 2011 14th International Conference. 2011: 569-572

[19] Yang Liu, Huang Jian-zhong, Xie Chang-sheng, et al. Raf: a random access first cache management to improve SSD-Based disk cache[C]// Networking, Architecture and Storage (NAS), 2010 IEEE Fifth International Conference. 2010: 492-500