网络存储阵列中 CACHE 的设计①

田新宇，马永强，王　伟

(西南交通大学 信息科学与技术学院，成都 610031)

摘　要：CACHE 是连接 CPU 与内存的一种高速缓冲存储器，用于提高系统的读写性能。本文中的 CACHE 正是借用了这个名词，而非真正的 CACHE，用内存模拟 CACHE 来实现高速的数据缓冲。

关键词：CACHE；线程；内存；高水位；低水位；CHUNK；Destage;

CACHE Design in Network Storage Array

TIAN Xin-Yu, MA Yong-Qiang, WANG Wei

(School of Information Science & Techology, Southwest Jiaotong University, Chengdu 610031, China)

Abstract: CACHE is a kind of high speed buffer storage device used to connect CPU and main memory, which can improve the speed of system's read and write. The referred CACHE in this article is not real CACHE, we use main memory to implement the function of CACHE.

Keywords: CACHE; thread; main memory; high waterlevel; low waterlevel; CHUNK Destage

1　CACHE在存储阵列中的作用

1.1 存储应用系统中的业务流程

图 1 是一种典型的存储应用系统中的写业务数据流程图。左边部分代表了数据在主机上的流向，然后通过 FC 协议或 iSCSI 协议或 SAS 协议传输到存储阵列，经过 FC 或 iSCSI 或 SAS 协议前端一直送到底层 SCSI 驱动，最后到达磁盘。

可以看到，在这个系统中 SCSI 协议是核心。TGT 模块在这里扮演的是双重目标器的角色。主机上的 SCSI 设备驱动代表的是 LUN 设备的驱动，也就是 SCSI 启动器，我们知道启动器和目标器是成对的概念，TGT 模块和目标器模块一起成为主机侧启动器的目标器。与此同时，存储阵列侧底层 SCSI 驱动模块也是一个启动器，当然，这个启动器在实现上和主机侧一样，可以是 iSCSI 启动器也可以是 SAS 和 FC 启动器，它们的目标器仍然是 TGT 模块。

从图上可以看到，存储阵列的作用实际上是把主机准备下盘的数据接管过来处理，由阵列来下盘。这里存在一个疑问：为什么加了这么多处理流程之后还能提高数据吞吐速度？在这里起关键作用的是 CACHE，它能把所有来不及处理的用户数据缓存起来，延迟处理，而这一切对用户是透明的。目前来说，对存储阵列性能影响最大的就是 CACHE 模块的性能。

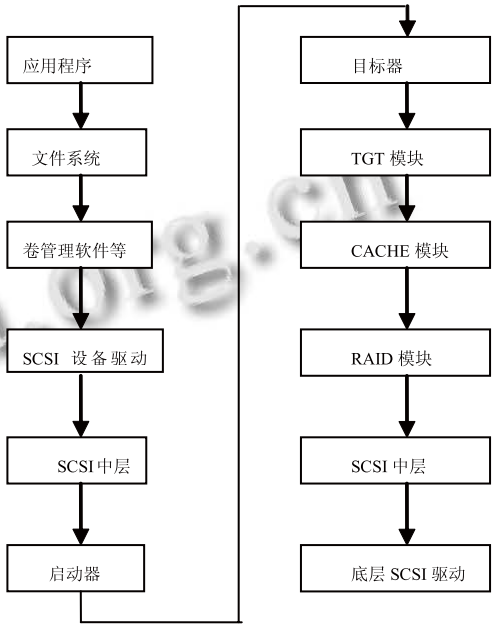

图 1　系统业务流程图

① 收稿时间:2010-09-27;收到修改稿时间:2010-11-01

1.2 回写与透写

回写是指将数据写入 CACHE 中就返回写成功，这时数据只写入 CACHE 而没有写入磁盘，因此称为脏数据，透写是指将数据写入 CACHE，再由 CACHE 交给 RAID 写入磁盘，待 RAID 模块返回后再将结果返回给上层模块。

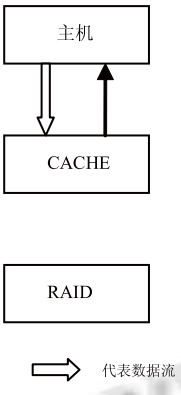

图 2 回写

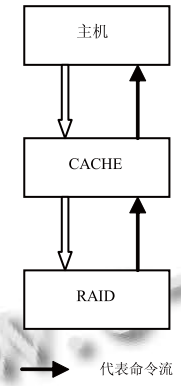

图 3 透写

1.3 CACHE 的主要作用

(1) 接受主机的写请求，CACHE 处于透写模式，则将数据写入内存，之后再将数据交给 RAID 模块写入硬盘，待 RAID 模块返回后再将返回结果返回给上层模块。

(2) 接受到主机写请求，CAHCE 处于回写模式，则将数据写入内存，然后向上层模块返回成功。

(3) 接受到主机写请求，CACHE 处于镜像写方式，则先将数据发送到对端写入内存，对端完成写入内存操作后，本端再通过回写方式将数据写入内存。

(4) 接受主机读请求，如果读命中，则直接将数据缓存在内存中的数据直接返回给上层模块。

(5) 如果读不命中，则通过 RAID 模块读取磁盘数据到内存中，再将数据返回给上层模块。

(6) 周期性进行检查和判断，淘汰只写入 CACHE 而没有写入磁盘的脏数据到磁盘，此过程称为 Destage。

2 CACHE的内存分布

底层驱动提供两个接口函数，用于获取系统的可以直接管理的内存区域的起始地址和大小，内存中的 1G 留给操作系统使用，剩余的内存就分配给 CACHE 模块使用。

图 4 中就是分配给 CACH 使用的内存的划分，预留给镜像通道 IBS 模块 1M，分配 63M 给为系统提供 REQ 和 SGL 资源的 Resource REQ-SGL 模块，剩下的内存中的 6%分配给数据组织子模块 CACHE ORG，94% 分配给为系统提供 4K 页面内存管理的 BUFFPOOL 子模块。

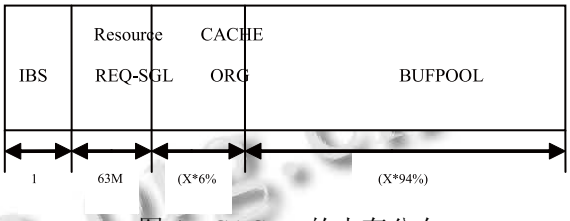

图 4 CACHE 的内存分布

3 CACHE的数据组织

REQ-SGL 的数据组织

REQ（request）是上层用来发送读写请求的一种数据结构，SGL（scatter gather list）是用来携带数据的一种数据结构，可以形象的打个比喻，SGL 是火车的车厢，而 REQ 就是火车头。数据装到车厢里面，再把车厢放到或者上，通过火车把数据一站一站的运送到目的地。

REQ 在设计上只能用于两个模块之间传递数据，在读写流程中是以 REQ 结构作为载体传递上层模块的操作请求,而 SGL 是上下层传递数据的媒介。从实现上可以看到，它包含了一个生产者和一个消费者队列。

```
typedef struct tagREQ_S{
OSP_U32 ulOpCode;     /*标识该请求的类型*/
OSP_U32 ulLun;        /*标识该请求要访问
Lun*/
OSP_U32  ulLba;   /*标识该请求要访问的起始扇区号*/
OSP_U32  ulLen;   /*标识该请求要访问的扇区数目*/
SGL_S*pSgl;         /*指向一个 SGL 结构，其每个页面都是本地页面*/
struct list_head producer_node;     /*用于把请求串入生产者的相应队列*/
struct list_head consumer_node;     /*用于把请求串入消费者的相应队列*/
void*pStorUp;       /*用于标识上一个请求*/
void(*done)(struct  tagREQ_S*);       /*请求的回调
```

函数*/

}REQ_S;

SGL 这个结构体是系统中通用的数据结构，用于数据传递。其中 SGL_ENTRY_S 结构体可以理解为页面指针。在设计上，这个结构体最大可以包含 800 个页面结构体。

typedef struct tagScatterList{

SGL_ENTRY_S astEntry[ENTRY_PER_SGL]; /*指向 SGL_ENTRY_S 类型数组*/

unsigned int ulEntrySum; /*用于标识 SGL_ENTRY_S 类型数组元素个数*/

unsigned int ulCurrentEntry; /*标识正在访问的 SGL_ENTRY_S 类型数组的元素*/

}SGL_S;

4 CACHE的读写请求的处理

4.1 CACHE 中读写请求的处理

CACHE 模块处理主机和 IBS（板间通道）传过来的 REQ 请求，根据 REQ 请求的不同类型，分别操作对应的内存空间，实现快速的读写操作。当上层 TGT 模块下发一个 REQ 时，CACHE 模块收到 REQ 后会激活一个分配线程，分配线程根据 REQ 中的请求类型将 REQ 分配给相应的线程，如果是读请求，则分配线程就将此 REQ 挂到读请求等待链表上，然后释放一个信号量，激活读线程，如果是写请求，则将此 REQ 挂到写请求等待链表上，然后释放一个信号量，激活写线程。然后由线程自己处理读写请求。

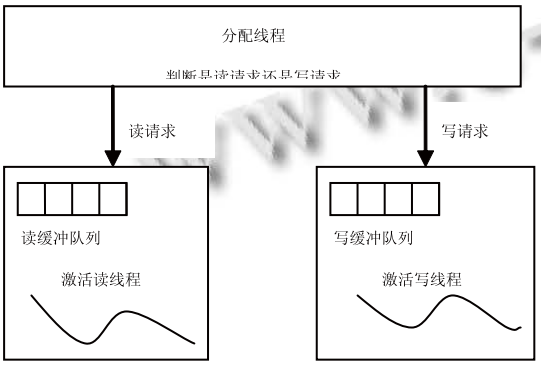

图 5　读写请求处理

4.2 读线程

当 CACHE 模块接受到 REQ 请求后，将 REQ 请求加入 g_pstReqWaitQueue 读写全局链表中，并触发 ReadWrite 线程去处理对应的 REQ 请求，在 ReadWrite 线程中的处理流程如下：

(1) 判断 g_pstReqWaitQueue 读写全局链表是否为空，如果读为空，则处理写冲突，写操作产生的冲突分两种：一写操作产生的冲突，二读操作产生的冲突。

(2) 如果链表不为空，则取出 REQ 节点并根据其操作类型进行分类处理。

(3) 如果 REQ 操作类型为读完成操作，表示读到的数据已经上传给主机，可以释放 REQ 和 SGL 资源了。

(4) 如果 REQ 为读操作，则先申请一个空的 SGL 悬挂在 REQ 上，并查看内存中是否存在要读的数据:

(ⅰ) 如果存在则认为读命中，并将数据串在 SGL 上，通知 SCSI 模块读到数据，同时根据智能预取的策略进行预取。

(ⅱ) 如果读不命中，判断是否在预读队列 s_astLargeReadPending 中，若在预读队列中也算读命中。

(ⅲ) 如果读不命中，则根据 LUN 的预读策略（固定预读、倍数预读、智能预读）创建一个预读请求挂到预读队列 s_astLargeReadPending 中，并为预读请求的 SGL 申请 4K 页面资源，在申请到页面资源后，将预读请求挂载到 g_astReadReqWaitQueue 链表上触发 READ 线程，通知块设备读取数据到预读请求的 SGL 中。

(ⅳ) 将预读到的数据写入内存中，同时释放预取请求，预读完成后重新执行悬挂在预读队列中的主机读请求，若命中则通知 SCSI 模块读到数据，若不命中则释放 SGL 资源并通知 SCSI 读失败。

要注意的是在 CACHE 的读操作过程中，所有的数据都是先存在内存中的，然后将数据串到 SGL 上返回给 SCSI 接口模块，而在内存中的数据是靠 CHUNK (一种数据单位，包含 16 至 64 个 4K 页面,4K 页面有 BUFFPOOL 模块来提供)来管理的，因此在读之前首先会对其引用计数，防止在将数据返回给 SCSI 模块过程中被其它线程将 CHUNK 中保存的数据给释放掉。

4.3 写线程

写操作分为回写、透写和镜像写三种类型，回写是将 REQ 带的数据写入本地内存之后就返回写成功，镜像写是将 REQ 带的数据写入对端控制器的镜像内

存区，再写入本地内存后才返回成功，透写是将 REQ 带的数据写入内存区，在写入磁盘后才返回成功，不管是何种写方式，其 REQ 都是在 SCSI 模块产生的，而 SCSI 模块在产生 REQ 时，也要同步申请 SGL 资源，并将要写的数据放入 SGL 指向的 4K 页面上，需要注意的是写操作类型不是在 SCSI 模块产生 REQ 时指定的，而是 LUN 的配置属性决定的，因此在改变 LUN 的配置属性时也会改变 REQ 的写操作类型。

4.4 Destage 线程处理

在回写的过程中会产生脏数据，而脏数据需要下发到磁盘中，对脏数据的刷盘操作是在 Destage 线程中完成的，触发 Destage 线程有如下几个操作，一、回写操作完成后会触发 Destage 线程刷盘，二、有一个 LUN 脏数据刷盘的命令请求，三、周期性的触发 Destage 线程刷盘。

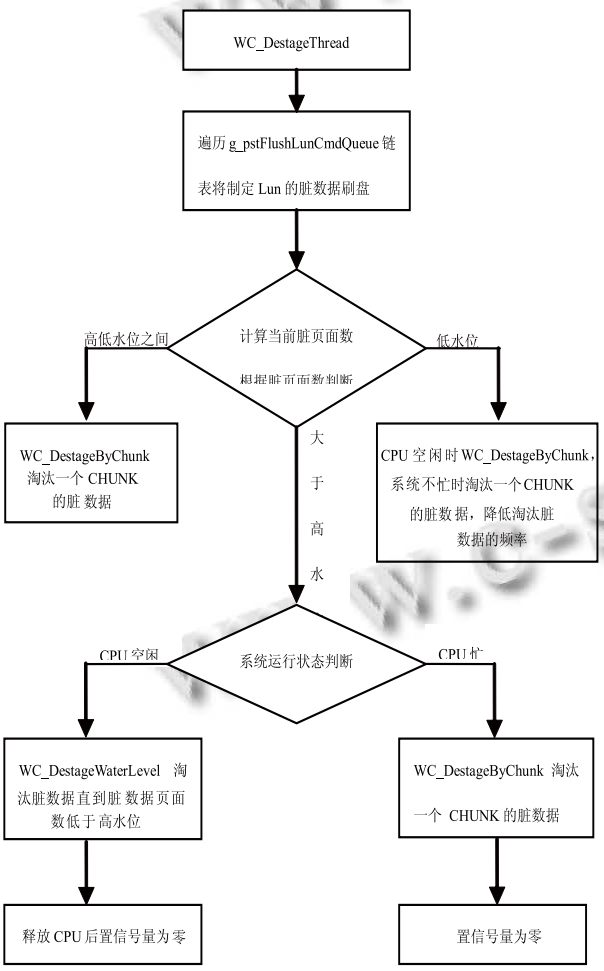

图 6 Destage 线程

Destage 处理过程如下：

(1) 存在将整个 LUN 的脏数据刷盘的请求，先将 LUN 中的脏数据刷盘。

(2) 根据脏页面数进行判断，若大于高水位，先判断系统运行状态，系统忙时只淘汰一个 CHUNK 的脏数据，系统闲时，淘汰 CHUNK 脏数据直到脏页面数低于高水位。

(3) 脏页面数介于高低水位之间，则只淘汰一个 CHUNK 的脏数据。

(4) 脏页面数低于低水位，则在系统不忙时判断 1 秒内是否有写 IO，如果没有则淘汰一个 CHUNK 脏数据，若系统忙时或 1 秒内还有写 IO 则不刷盘。

脏数据刷盘时会将 CHUNK 数据迁移到写队列中，因此在刷盘失败后要将 CHUNK 数据从写队列中迁移回来，以备后续可以继续进行刷盘处理。

5 CACHE对提高阵列性能的作用

本测试使用 IOMeter 分别在透写（相当于没有 CACHE 的功能）和回写（加入 CACHE 的功能）的条件下对阵列下 IO，配置：IO 大小 4k，50%顺序读，50%顺序写，运行 10 分钟，采样 500 次。

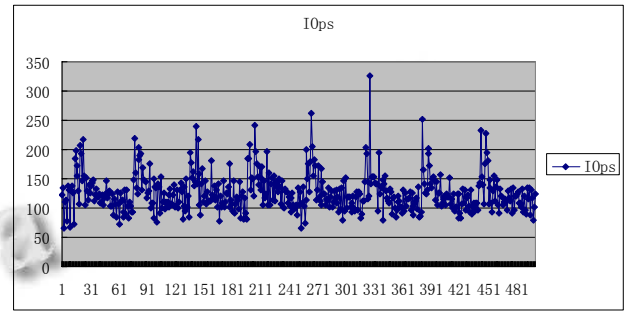

图 7 透写条件下的 IO 统计

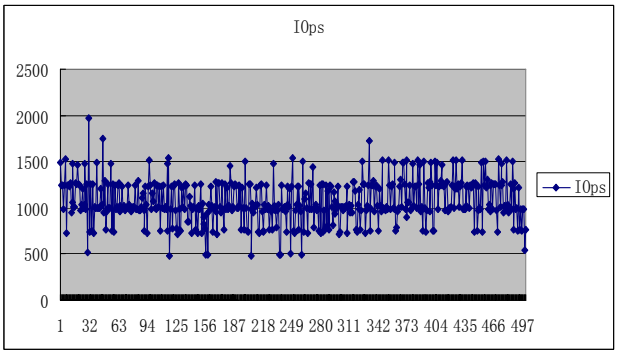

图 8 回写条件下的 IO 数统计

(下转第 150 页)

对 foreman.yuv 的编码时间进行比较，序列类型为 IPPP，共 3 帧，帧率为 30f/s，量化参数 QP=28，使用 Hadamard 变换和 CAVLC 熵编码。未加密的编码总时间为 2.296s，运动估计总时间为 0.627s；进行混沌加密的编码总时间为 2.329s，运动估计总时间为 0.663s，加密效率较好。

5 结语

本文通过 Logistic 映射产生用于加密的混沌序列，然后对 H.264 的 CAVLC 熵编码阶段进行加密。相对于完全加密算法和部分加密算法，计算复杂度低；相对于 DCT 系数加密算法，没有改变压缩比。实验结果表明，该加密系统不改变视频码流的格式，加密速度快，对密钥非常敏感，并且具有很大的密钥空间，对各种攻击具有较强的抵抗性，具有良好的应用价值。

参考文献

1 Qao L, Nahrstedt K. A new algorithm for MPEG video encryption. Proc. of the First International Conference on Imaging Science, Systems and Technology (CISST'97). LasVegas, Nevada, 1997:21–29.

2 Chiaraluce F, Ciccarelli L, Gambi E, Pierleoni P, Reginelli M. A new chaotic algorithm for video encryption. IEEE Trans. on Consumer Electronics, 2002,48(4):838–844.

3 廉士国,孙金生,王执铨.集中典型视频加密算法的性能评价.中国图像图形学报,2004,9(4):483–490.

4 Tang L. Methods for encrypting and decrypting MPEG video data efficiently. Proc. of the Fourth ACM International Multimedia Conference(ACM Multimedia'96). Boston, MA, 1996:219–230.

5 Tosun AS, Feng WC. Efficient multi-layer coding and encryption of MPEG video streams. IEEE International Conference on Multimedia and Expo. New York, 2000, 1:119–122.

6 Draft ITU-T recommendation and final draft international standard of joint video specification(ITU-T Rec. H.264/ISO/IEC 14 496-10 AVC. In Joint Video Team(JVT) of ISO/IEC MPEG and ITU-T VCEG, JVTG050, 2003).

7 毕厚杰.新一代视频压缩编码标准:H.264/AVC.北京:人民邮电出版社,2005.

8 李晓举,冯战申,胡友情.基于 H.264 CAVLC 熵编码的视频加密方案.计算机工程与应用,2009,45(34):114–117.

9 于志宏,王静波,刘喆,等.基于 Logistic 和 Baker 映射的视频加密方法.吉林大学学报,2008,26(3):253–258.

10 严三国,陈永彬.Logistic 满映射混沌序列性能分析.电子技术,2009:194–196.

11 张波.混沌 PN 序列的性能分析与优化设计[硕士学位论文].杭州:杭州电子科技大学 CAD 研究所,2009.25–28.

(上接第 191 页)

从统计数据可以看出，回写条件下性能比透写提高了 10 倍左右，由此可见，CACHE 极大的提高了系统的性能，同时 IO 的波动范围也都在 20%左右，系统运行比较稳定。

6 总结

(1) 文中的 CACHE 设计采用多线程，多个线程可以并行处理，极大的节约了读写 IO 的时间，提高了 CACHE 和存储阵列的整体性能。

(2) 同时设置了高低水位，根据系统实现情况来判断是否刷脏数据，提高性能的同时也提高了系统的稳定性。

参考文献

1 Chen TF, Baer J. Eeffetive hardware-based data prefetching for high-performance processors. IEEE Trans. on Computers, 1995,44(5).

2 Hsu WC, Smith JE. A perofrmance study of instruction cache prefetching methods. IEEE Trans. on Computers, 1998, 47(5):497–508.

3 Stallings W.计算机组织与结构—性能设计.第 5 版.张昆藏等译.北京:电子工业出版社,2001.

4 汤子瀛.计算机操作系统.西安:西安电子科技大学出版社,1994.