第 36 卷　第 14 期　　计　算　机　工　程　　2010 年 7 月
Vol.36　No.14　　Computer Engineering　　July 2010
·软件技术与数据库·　文章编号：1000—3428(2010)14—0056—03　文献标识码：A　中图分类号：TP311.12

基于 EVMS 的带外虚拟存储系统结构

林伟兵[1]，赵跃龙[1]，王文丰[2]，陈　超[1]
(1. 华南理工大学计算机科学与工程学院，广州 510640；2. 南昌工程学院计算机科学与技术系，南昌 330099)

摘　要：分析当前网络存储系统存在的问题，给出一种基于企业卷管理系统技术的带外虚拟网络存储系统基本结构。分别阐述该存储系统的客户端、内部数据的读/写操作、存储系统在线存储容量扩展、负载均衡和元数据异步更新等算法的设计思想与实现方法。
关键词：网络存储；企业卷管理系统技术；带外；在线容量扩展；负载平衡

Structure of Out-of-band Virtual Storage System Based on EVMS

LIN Wei-bing[1], ZHAO Yue-long[1], WANG Wen-feng[2], CHEN Chao[1]
(1. School of Computer Science and Engineering, South China University of Technology, Guangzhou 510640;
2. Department of Computer Science and Technology, Nanchang Institute of Technology, Nanchang 330099)

【Abstract】This paper analyzes some problems of existing network storage system and presents a basic architecture for out-of-band virtual network storage system based on Enterprise Volume Management System(EVMS) technology. The client, the internal read and write operation, the online expansion of storage capacity, the load balancing, and the asynchronous metadata update algorithms for this out-of-band virtual storage system are designed and implemented.
【Key words】network storage; Enterprise Volume Management System(EVMS) technology; out-of-band; online capacity expansion; load balance

1　概述

随着计算机和网络技术的高速发展，海量的数据对现有存储技术提出了更高要求。目前较成熟的网络存储系统有直接存储系统(Direct Attached Storage, DAS)、联网存储系统(Network Attached Storage, NAS)和存储区域网络(Storage Area Network, SAN)[1]。但大量以数据为中心的网络应用的出现，对存储系统在超大容量、高可靠性、高性能、在线容量扩展、存储空间按需分配等方面的表现提出了更多要求。

SAN 存储虚拟化技术的出现较好地满足了上述需求。依据 SAN 中数据通路和命令通路的特点，SAN 存储网络级虚拟化可以分为带内(in-band)和带外(out-of-band)2 种方式。带内方式实现简单，但容易造成性能瓶颈[2]。带外方式通过分离数据传输和元数据访问来解决该问题。目前较成熟的带外虚拟化系统有 EMC 公司的 Invista、清华大学的 TransCom[3]等，但这些系统在元数据服务器访问性能、存储空间的按需分配、存储设备负载和存储系统的在线扩展能力等方面通常存在问题，很难满足网络上不断增长的海量和高性能的存储需求。因此，研究并实现新型虚拟化网络存储系统具有重要意义。

2　基于 EVMS 的带外虚拟网络存储系统

2.1　EVMS 简介

企业卷管理系统(Enterprise Volume Management System, EVMS)[4]是 IBM 开发的企业级逻辑卷管理器，主要由 UI、Engine 和 Plug-in 3 个部分组成，如图 1 所示。UI 负责给用户提供一个直观的操作接口，Plug-in 实现各个插件的管理功能，Engine 对 Plug-in 的各个插件功能进行抽象，提供统一的接口供 UI 调用。

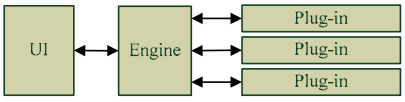

图 1　EVMS 的架构

2.2　系统结构

针对目前带外虚拟网络存储系统存在的问题，本文给出一种基于 EVMS 的带外虚拟网络存储系统结构，如图 2 所示。

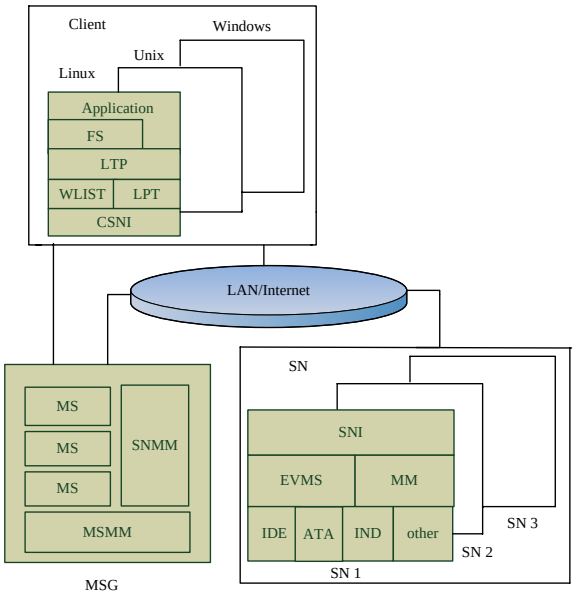

图 2　基于 EVMS 的带外虚拟网络存储系统结构

从图 2 可以看出，整个系统由客户端(Client)、LAN/

基金项目：国家自然科学基金资助项目(60573145)；教育部博士点基金资助项目(200805610019)；广州市科技计划基金资助项目(2007J1-C0401)
作者简介：林伟兵(1984－)，男，硕士研究生，主研方向：网络存储技术；赵跃龙，教授、博士生导师；王文丰，讲师、博士；陈　超，硕士研究生
收稿日期：2010-02-19　**E-mail：**linweibing2006@163.com

Internet、元数据服务器组(Metadata Server Group, MSG)和存储节点(Storage Node, SN)4 个部分组成。

Client 负责将上层应用程序或文件系统的 I/O 请求通过 LPT 表重定向到存储系统的 SN 上，当客户端缺少可写的空闲数据块时，它负责向 MSG 申请空闲的数据块。

MSG 对整个系统的元数据进行管理，主要实现如下功能：(1)通过 SNMM(Storage Node Management Model)对 SN 进行管理，使整个系统具有很强的扩展性并实现 SN 负载均衡；(2)在整个 MSG 中，采用多 MS(Metadata Server)的方式来处理客户端的请求，使各个客户端的请求能得到及时响应，并对各个 MS 中存储的客户端元数据进行“模备份”到另一个 MS 中，保证各个客户端元数据的可靠性；(3)通过 MSMM (Metadata Server Management Model)对各个 MS 进行有效调节，达到整个系统的 MS 负载均衡。

SN 的设计充分考虑了存储容量的在线扩展和存储设备的无关性问题，利用 EVMS 技术的特性，将底层存储设备(如 IDE、IND[5]等)虚拟化成连续的逻辑数据块。

3 系统工作原理

系统的主要数据结构如图 3 所示。在 SNMM 中，存储节点描述结构(Storage Node describe Structure, SNS)记录每个存储节点的信息和该节点的数据块使用情况(以位图 bitmap 表示)。在 MS 中，客户端使用存储系统资源描述结构(Storage System Resource Describe Structure, SSRDS)包含每个客户端使用存储系统资源的信息以及客户端逻辑地址与存储系统物理地址转换表(Logic address transform Physical address Table, LPT)。当客户端写请求且客户端没有空数据块时，客户端向存储系统申请写数据申请空间组(Write Data Application Capacity Group, WDACG)，该结构记录客户端逻辑地址与存储系统物理地址转换关系和写标识 TB(1 表示写、0 表示空)。

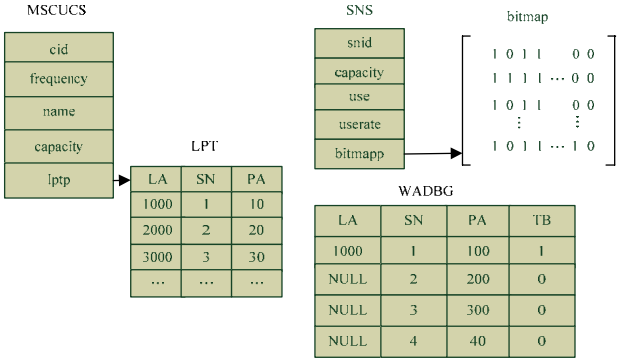

图 3 系统的主要数据结构

3.1 存储空间按需分配

为了解决存储空间按需分配的问题，本文采用预先分配逻辑地址空间的方法，在客户端的虚拟代理[3]在文件系统初始化时，设定文件系统访问的存储容量为无限大(如 1 PB)，此时设定的只是逻辑地址，只有当客户端进行写数据时，才向实际的存储系统申请物理空间，然后把客户端的逻辑地址与存储系统的物理地址通过 LTP 表对应起来，实现数据块的实际存储，以提高整个系统的存储空间利用率。

3.2 读/写操作算法

读/写操作对整个存储系统性能的影响极大，系统采用客户端依据 LPT 表、通过网络直接读存储系统的 SN 的方法，从而避免通过 MSG 来读数据块，减少读数据时系统的响应时间。在查找 LPT 时为了加快查找速度，对 LPT 表按客户端的逻辑地址进行排序并采用折半查找的方法来实现，具体操作如下：

算法 1

```
index = BSearch(LPT,readaddress);//折半查找
if(index != 0 && LPT[index].TB == 1){
read(LPT[index].SN,LPT[index].PA);//读数据块
return datablock;
}else
return 0;
```

客户端进行写数据块时，如果每写一个数据块便向 MSG 申请一个空闲数据块，那么每次写数据块时都要向 MSG 申请空闲数据块，将导致系统写数据性能差。为了解决该问题，本文设计“一次申请多次写”方式，即客户端向 MSG 申请空闲数据块时，都以 WDACG 为单位，假如每次客户端申请一个 WDACG 有 n 个空闲数据块，与每次申请一个数据块相比，减少 n-1 次通过网络向 MS 申请数据块的操作，明显提高写数据块的性能。且客户端可以根据写操作的频度来动态设定每次申请 WDACG 的数据块数 n 的大小，从而达到写性能和系统的存储空间利用率的平衡，具体操作如下：

算法 2

```
if(ishave(WDACG)){
getitem(WDACG);//取空地址对应关系项
writeLA(item,writeaddress);//对对应关系项写数据
modifyTB(item);//修改对应关系项的标识
insertsort(LPT);//插入 LPT 表，并排序
write(item.SN,item.PA,datablock);//把数据块写入存储系统
}else{
addwlist(WLIST);//加入写等待队列
apply(WDACG);//申请空闲数据块组
while(ishave(WLIST)){
get(WLIST);//取数据块
getitem(WDACG);
writeLA(item.writeaddress);
modifyTB(item);
insertsort(LPT);
write(item.SN,item.PA,item.datablock);
}
}
```

3.3 在线存储系统容量扩展

网络存储系统的在线容量扩展能力直接影响该存储系统的生命周期，是存储系统设计的一个关键问题。系统中设计了 2 种存储容量在线扩展方式，即扩展 SN 和扩展 SN 内部容量。EVMS 是以插件形式组成的企业级逻辑卷管理器，可以根据具体需要来写插件添加到 EVMS 中，实现存储设备的无关性，使系统容量扩展具有灵活性和高效性。

当存储系统需要大规模扩展存储容量(如合并原有的存储系统)时，可以新构造 SN 来扩展存储系统的容量，即扩展 SN，具体操作如下：

算法 3

```
create(SNS);//创建 SNS
init(SNS);//初始化 SNS
update(SNMM);//更新 SNMM
```

当有小容量的存储设备(如 ATA)要加入系统时，由于存储设备的容量较小，因此不必以 SN 的方式加入系统，根据 EVMS 的特性，可以把小容量的存储器加入到系统中，即扩展 SN 内部容量，具体操作如下：

算法 4

EVMS init;//启动 EVMS

add(device);//存储设备添加到 SN 中

update(SNS.bitmap);//更新 SN 的位矢图

update(SNS);//更新 SN

update(SNMM);//更新 SNMM

3.4 负载平衡算法

传统的带外虚拟网络存储系统只有一个 MS，当有多个客户端请求存储系统服务时，MS 容易成为系统的性能瓶颈，单个 MS 也会产生单点故障问题。为了解决上述问题，系统中采用 MSG 管理多个 MS 的方式。当有新的客户端加入存储系统时，MSG 的相应操作如下：

算法 5

设 N 是 MS 中客户端个数，f_i 是第 i 个客户端申请空闲空间的频度，F_j 是第 j 个 MS 中所有客户端申请存储空间的平均频度。

$F_j=\sum_{1}^{N} f_i/N$;//计算每个 MS 中的 F_j

$F_{min}=\min(F_1,F_2,\cdots,F_n)$;//选取平均频度最小的 MS

add($MS_{F_{min}}$);//把客户端加入平均频度最小的 MS

create(SSRDS);//创建 SSRDS

MSG 的形式比较灵活，当大规模的客户端加入存储系统时，MSMM 根据实时的客户端情况申请添加 MS，来解决 MS 过载问题。元数据对客户端和存储系统都极为重要，为了保证各个客户端元数据的可靠性，采用“模备份”方式将一个 MS 上的元数据备份到另一个 MS 中，当其中一个 MS 出现故障时，可以通过恢复找到故障 MS 中的元数据，提高了系统可靠性，具体操作如下：

算法 6

receive($WDACG_i$);//客户端 i 的元数据

findupdate(MS_i);//找到客户端对应的 MS_i，更新其客户端信息

$MS_{copy} = MS_i \mod N + 1$;//计算备份的 MS

copy(MS_{copy});//$WDACG_i$ 数据备份到 MS_{copy}

存储系统在考虑 MSG 负载均衡的同时也考虑了部分 SN 过载的问题，如果一个 SN 同时有大量客户端在访问，则必然会使这个节点处于繁忙状态，读/写性能将受到严重影响，且 SN 会因为过载而产生故障。为了防止部分 SN 过载，在每次客户端申请 WDACG 时都要通过 $UR_{min}=\min(UR_1,UR_2,\cdots,UR_n)$ 计算来选取使用率最小的 SN 为客户端分配空闲数据块，其中，UR_i 是第 i 个 SN 的使用率；n 是存储节点的个数。

3.5 元数据异步更新

整个存储系统在客户端与 MS 中各存在一个 LPT，为了达到这 2 个 LPT 的数据一致性和元数据服务器的传输信息和处理信息的压力较小，需要采用异步组更新元数据的方法。

当客户端的 WDACG 被写满时，客户端把 WDACG 信息发送给相应的 MS，MS 接收到客户端发送过来的 WDACG 后，更新对应的 LPT 表并进行元数据模备份，确保不会影响客户端读/写的性能。

4 结束语

本文提出的存储系统结构解决了带外虚拟网络存储的按需分配存储空间、高性能的读/写、在线存储容量扩展、负载平衡、元数据可靠性和元数据更新等问题，目前，基于 EVMS 技术的带外虚拟网络存储系统原型处于研制阶段，而现有存储系统存在一个问题，即一个客户端或多个客户端重复存储相同的数据块。重复存储相同的数据块降低了存储系统的性能和空间利用率，笔者考虑采用重复数据删除技术处理存储系统中数据块重复的问题，而有关存储系统的容错、备份、数据迁移和数据安全等问题有待进一步讨论并解决。

参考文献

[1] 赵文辉, 徐 俊, 周加林, 等. 网络存储技术[M]. 北京: 清华大学出版社, 2005.

[2] 王 迪, 舒继武, 薛 巍, 等. 块级别的海量存储虚拟化系统[J]. 清华大学学报: 自然科学版, 2007, 47(1): 108-111.

[3] 韦 理, 周悦芝, 夏 楠. 用于网络存储系统的存储空间动态分配方法[J]. 计算机工程, 2008, 34(5): 33-35.

[4] Enterprise Volume Management System(EVMS)[Z]. (2003-05-06). http://evms.sourceforge.net/.

[5] 赵跃龙, 戴祖雄, 王志刚, 等. 一种智能网络磁盘系统结构[J]. 计算机学报, 2008, 31(5): 858-867.

编辑 陈 晖

(上接第 55 页)

其中，测试用例的 4 个数字分别为(a, n, x, y)。这 4 个数字是需要用户输入的变量值。通过测试用例 $T1$ 和 $T2$，可以得到关键谓词为 $S12$。对于错误输出的测试用例 $T2$，$S12$ 后向切片为{$S4$, $S9$, $S11$, $S12$}，错误输出语句 $S15$ 后向数据切片为{$S4$, $S9$, $S13$, $S15$}。两切片的并集 St 为{$S4$, $S9$, $S11$, $S12$, $S13$, $S15$}。对于测试用例 $T3$，求出执行语句 E 和 St 的交集{$S4$, $S9$, $S11$, $S12$, $S13$, $S15$}。该集合含有错误的可能性很高，且比原程序集合小。通过进一步分析，可以确定错误发生在语句 $S11$。

表 1 测试用例列表

编号	测试用例	*PSet*()	*DSet*()	错误数	正确数
*T*1	(8, 1, 2, -2)	*S*3, *S*7, *S*10, *S*12	*S*4, *S*9, *S*14, *S*15	5	5
*T*2	(6, 1, 2, 2)	*S*3, *S*7, *S*10, *S*12	*S*4, *S*9, *S*13, *S*15	3	1
*T*3	(8, 1, 4, 2)	*S*3, *S*7, *S*10, *S*12	*S*4, *S*9, *S*13, *S*15	5	3

4 结束语

本文提出一种基于关键谓词并同时考虑代码优先的算法。该方法求出 2 个切片的并集，并按照代码优先的思想将错误输出语句的测试用例执行路径与此并集相比，得到需要测试的集合。它减少了分析语句的数量，得到的语句集合包含错误的可能性极大增加，降低了错误定位难度。

本文中确定关键谓词的方法在多重谓词结构中适用性不高，这是下一步工作需要研究的内容。

参考文献

[1] Weiser M. Programmers Use Slices When Debugging[J]. Communications of the ACM, 1982, 25(7): 446-452.

[2] 李必信. 一种分析和理解程序的方法——程序切片[J]. 计算机研究与发展, 2000, 37(3): 284-291.

[3] Zhang Xiangyu, He Haifeng. Experimental Evaluation of Using Dynamic Slices for Fault Location[C]//Proc. of the 6th International Symposium on Automated and Analysis-driven Debugging. California, USA: [s. n.], 2005.

[4] Sun Jirong, Li Zhshu, Ni Jianchen, et al. Software Fault Localization Based on Testing Requirement and Program Slice[C]//Proc. of International Conference on Networking, Architecture and Storage. Guilin, China: [s. n.], 2007.

[5] Zhang Xiangyu, Gupta N. Locating Faults Through Automated Predicate Switching[C]//Proc. of the 28th International Conference on Software Engineering. Shanghai, China: [s. n.], 2006.

编辑 陈 晖