文章编号:1001 - 9081(2006)11 - 2756 - 03

基于 Reed-Solomon 算法的 RAID 机制的设计与实现

赵 伟[1],莫国庆[1],那宝玉[1],刘 鹏[2]
(1. 解放军理工大学 指挥自动化学院,江苏 南京 210007;
2. 解放军理工大学 全军军事网格技术研究中心,江苏 南京 210007)
(joypla@gmail.com)

摘 要:为了满足海量信息存储可靠性的要求,提出了把 Reed-Solomon 算法应用到 RAID 系统中的方法,并给出了在 Linux 环境下系统实现的方案 RSRAID。通过对系统性能及可靠性进行测试,并与其他 RAID 机制进行对比,证明系统具有良好的 I/O 性能和更高的可靠性。

关键词:海量存储;高可靠性;Reed-Solomon 算法;RSRAID;Linux
中图分类号: TP4264 **文献标识码**:A

Design and implementation of RAID mechanism based on Reed-Solomon algorithm

ZHAO Wei[1], MO Guo-qing[1], NA Bao-yu[1], LIU Peng[2]
(1. *Institute of Command Automation, PLA University of Science and Technology, Nanjing Jiangsu* 210007, *China*;
2. *PLA Research Center for Military Grid Technology, PLA University of Science and Technology, Nanjing Jiangsu* 210007, *China*)

Abstract: To satisfy the reliability of mass information storage, a method to apply Reed-Solomon algorithm to RAID system was proposed, and a realization under Linux system named RSRAID was given. Experiment results prove that RSRAID has good I/O performance and better reliability compared with other RAID systems.

Key words: mass storage; high reliability; Reed-Solomon algorithm; RSRAID; Linux

0 引言

独立冗余磁盘阵列(Redundant Array of Independent Disks,RAID)[1]是由多个小容量、独立的硬盘组成的阵列,而阵列综合性能可以超过单一昂贵的大容量硬盘的性能。由于是对多个磁盘并行操作,所以 RAID 磁盘子系统与单一磁盘相比输入输出性能得到了提高。服务器把 RAID 阵列看成一个单一的存储单元,并对几个磁盘同时访问,所以提高了输入输出的速率。RAID 还具有单磁盘不具有的优点,它可以通过数据校验,提供容错功能。RAID 又分为 0 到 5 等几个级别,以上几种 RAID 的级别最多只能允许磁盘阵列中的一个崩溃,如果一个以上的磁盘同时崩溃,数据就无法恢复了。为了提高系统的顽存性和抗毁性,必须增加校验盘的数量,而 Reed-Solomon 算法就能够满足这种要求。本文介绍的方法可以通过 Reed-Solomon 算法在理论上容忍 $M(M \geqslant 1)$ 个磁盘的崩溃,从而实现一种高可靠性的存储系统。

1 Reed-Solomon 算法在 RAID 中的应用

Reed-Solomon 算法[2]是一种前向错误校正算法,通常用于数据传输和存储应用,通过在数据传输和存储之前增加冗余,Reed-Solomon 编码/解码器可以检测和校正数据块的错误。不同于此算法在通信中的应用,此算法在 RAID 系统中的作用是恢复“擦除”,当一个设备崩溃时,它就会关闭,而且系统可以察觉到这种行为。这不同于存储和读取中出现的“错误”,这种错误只能通过内嵌的编码才能发现[3]。为了适应其在 RAID 中的应用,需要对其进行改进:

用 Di 表示第 i 个数据盘,用 Ci 表示第 i 个校验盘,为了计算出 Ci 中的内容,需要一个函数 Fi 对所有的数据盘进行校验计算。如果把这种算法称作 RSRAID 的话,那么 $RSRAID(n, m)$ 就表示共有 n 个数据盘, m 个校验盘;而算法的实施又是基于 words 的,每个磁盘被划分为若干个 words,每个 words 的大小是 w bits, w 的大小可以根据具体的系统来定,这样,每个磁盘就包含 $l = (\text{kbytes})\left(\frac{8\text{bits}}{\text{byte}}\right)\left(\frac{1\text{word}}{w\ \text{bits}}\right) = \frac{8k}{w}$ words,那么计算的过程如图 1 所示。

$d_{1,1}$	$d_{2,1}$	$c_{1,1}=F_1(d_{1,1},d_{2,1})$	$c_{2,1}=F_2(d_{1,1},d_{2,1})$
$d_{1,2}$	$d_{2,2}$	$c_{1,2}=F_1(d_{1,2},d_{2,2})$	$c_{2,2}=F_2(d_{1,2},d_{2,2})$
$d_{1,3}$	$d_{2,3}$	$c_{1,3}=F_1(d_{1,3},d_{2,3})$	$c_{2,3}=F_2(d_{1,3},d_{2,3})$
⋮	⋮	⋮	⋮
$d_{1,l}$	$d_{2,l}$	$c_{1,l}=F_1(d_{1,l},d_{2,l})$	$c_{2,l}=F_2(d_{1,l},d_{2,l})$

图 1 $n = 4, m = 2$ 时的计算过程

图中的 $d_{i,j}$ 表示第 i 个磁盘的第 j 个 word。计算公式为: $c_i = F_i(d_1, d_2, \cdots, d_n) = \sum_{j=1}^{n} d_j f_{i,j}$,如果用 $\boldsymbol{D}$ 表示数据盘,$\boldsymbol{C}$ 表示校验盘,上式可以表示为:$\boldsymbol{FD} = \boldsymbol{C}$,定义 $\boldsymbol{F}$ 为 $m \times n$ 阶的范德蒙德矩阵,其中 $f_{i,j} = j^{i-1}$,所以 $\boldsymbol{FD} = \boldsymbol{C}$ 可以表示为以下矩阵:

$$\begin{bmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,n} \\ f_{1,1} & f_{1,2} & \cdots & f_{1,n} \\ \vdots & \vdots & & \vdots \\ f_{m,1} & f_{m,2} & \cdots & f_{m,n} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

如果磁盘阵列中有 $j \leqslant m$ 个磁盘崩溃,根据下面的方法进

收稿日期:2006 - 05 - 23;**修订日期**:2006 - 07 - 03 **基金项目**:国家自然科学基金资助项目(60403043)

作者简介:赵伟(1981 -),男,河北景县人,硕士研究生,主要研究方向:网络存储、Linux 内核、分布式系统; 莫国庆(1965 -),男,浙江湖州人,副教授,硕士,主要研究方向:系统软件、多媒体技术; 那宝玉(1979 -),男,黑龙江阿城人,博士研究生,主要研究方向:网络存储、安全操作系统; 刘鹏(1970 -),男,四川绵阳人,副教授,硕士,主要研究方向:并行/分布体系结构、网格计算.

行数据恢复:构建一个$(m+n)\times n$阶矩阵 $\boldsymbol{B}=$

$$\begin{bmatrix} 0^0 & 0^1 & 0^2 & \cdots & 0^{n-1} \\ 1^0 & 1^1 & 1^2 & \cdots & 1^{n-1} \\ 2^0 & 2^1 & 2^2 & \cdots & 2^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ (n+m-1)^0 & (n+m-1)^1 & (n+m-1)^2 & \cdots & (n+m-1)^{n-1} \end{bmatrix};$$

前 n 行是一个单位矩阵,删除掉任意 m 行后的矩阵为一个可逆矩阵,这样当一块磁盘崩溃后,就在 $\boldsymbol{B}$ 和 $\boldsymbol{C}$ 中删去对应的一行,然后使用矩阵的基本变换把前 n 行变为单位矩阵,得到矩阵 $\boldsymbol{B}'$,有 $\boldsymbol{B}'\boldsymbol{D}=\boldsymbol{C}'$,这样可以通过高斯消去法得到 $\boldsymbol{D}$,也就恢复了数据盘中的内容,如果有校验盘损坏的话,可以重新根据 Fi 计算出校验盘中的数据。

2　系统设计方案

系统的试验平台选择了 Linux,内核版本为 2.6.11。Linux 有开源的优势,并且 Linux2.6.11 内核支持软件 RAID 机制,通过修改 Linux 内核驱动 MD 部分的源代码,重新编译生成支持 RSRAID 机制的内核,部署在网络上的某台主机上,就实现了本文的设计目标。系统的层次模型如图 2 所示。

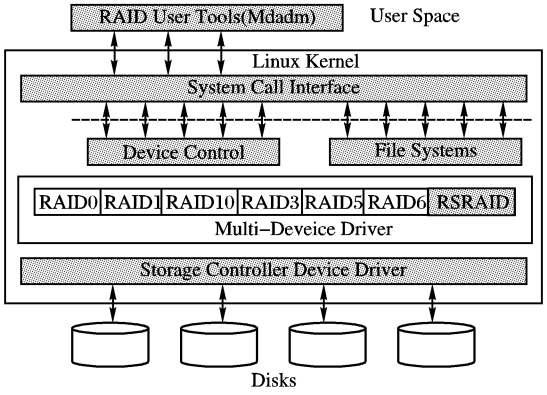

图 2　嵌入了 RSRAID 机制的 Linux 内核层次模型

Linux 中的 RAID 机制是由内核中的 MD 部分[4]实现的,Linux2.6.11 已经实现了 RAID0,RAID1,RAID5,RAID6 等几种机制,为了实现 $M(M\geqslant 1)$ 个校验磁盘的 RAID 机制,需要实现一种新的 RAID 机制。参照 RAID6 在 RAID5 机制上的扩展,我们设计了 RSRAID 机制,主要工作是把改进后的 Reed-Solomon 算法引入 RAID 机制。在以往的 RAID 机制中,产生校验码的算法主要是借助于 Linux 中的 XOR 库函数,也就是通过简单的异或,这种方法限制了校验盘的数量,而改进 Reed-Solomon 算法在理论上可以实现 M 个校验盘数据的校验计算。具体的实现中就要把改进 Reed-Solomon 算法与 RAID 机制有机结合起来。具体流程如下:

1)系统初始化时,也就是用户开始建立一个 RAID 阵列时,需要根据用户设置的参数来初始化 RS 算法的参数(本文的后半部分,Reed-Solomon 算法简称为 RS 算法),rs_encoder = init_rs (n , 0x11d, 1, 1, m);其中 n 代表 RAID 阵列磁盘总的数目,m 代表校验盘的数目,其他几个参数是 RS 算法固有的参数。初始化函数 init_rs 返回一个指针 rs_encoder,以后的编解码函数都要根据这个指针来获取阵列的基本信息。

2)初始化完成后,阵列的 build 过程开始,这个时候,RS 算法的 encode 函数被调用,用来计算阵列的校验数据,计算完成后,RAID 阵列就可以进入运行状态。

3)当阵列中的数据更新时,校验数据也要随之更新,更新的方式是使用 encode 函数重新计算校验数据。

4)当阵列中有不多于 $M(M=m)$ 的磁盘崩溃时,系统进入降级运行模式,当有可用的空闲磁盘时系统开始进入恢复状态。此时 decode 函数被调用来计算出损失的数据。

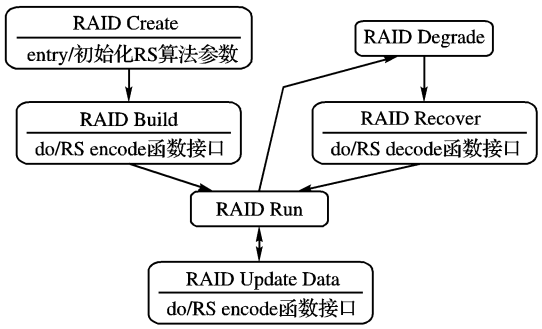

图 3　RS 算法在 RAID 机制中的运用流程

3　系统可靠性分析

从可靠性理论来看,RSRAID 属于马尔可夫串联型的可修复系统[5]。为了分析系统的可靠性,做如下假定:1)每个磁盘的失效率 λ 是一个常数,即 $\lambda=1/MTBF_{disk}$;2)每个磁盘的修复率也是一个常数,即 μ 也是一个常数,$\mu=1/MTTR_{disk}$($MTTR_{disk}$ 为单个磁盘的平均修复时间,指磁盘驱动器从失效到恢复正常工作的时间,主要由磁盘替换时间和数据重构时间组成);3)失效事件和修复事件彼此互相独立。

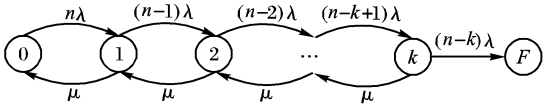

图 4　k 个盘失效时的马尔可夫可靠性模型

k 个盘失效且系统可恢复时 RSRAID 的可靠性及其数学模型的分析如下:在图 4 中,状态“0”表示组内每个盘都正常工作,状态“1”表示有一个盘失效,状态“2”表示有两个盘失效……状态“k”表示有 k 个盘失效。由于采用了冗余容错技术,所以直至状态“k”失效时,系统仍可以正常工作。状态“$k+1$”为系统无法正常工作状态。由马尔可夫模型,可以得到如下方程组:

$$\begin{aligned} P_0(t) &= -n\lambda P_0(t)+\mu P_1(t) \\ P_1(t) &= n\lambda P_0(t)-[\mu+(n-1)\lambda]P_1(t)+\mu P_2(t) \\ P_2(t) &= (n-1)\lambda P_1(t)-[\mu+(n-2)\lambda]P_2(t)+\mu P_3(t) \\ &\vdots \\ P_k(t) &= (n-k+1)\lambda P_{k-1}(t)-[\mu+(n-k)\lambda]P_k(t) \end{aligned}$$

设初始条件:$P_0(0)=1$, $p_1(0)=p_2(0)=\cdots=p_k(0)=0$。将上面的方程组进行拉氏变换最终得到系统的可靠度为:

$$MTTF_{RSRAID}=\sum_{j=0}^{k}\frac{(-1)^{j-1}\{[\mu+(n-k)\lambda]\Phi_{k-1}^{(i)}-\mu(n-k+1)\lambda\Phi_{k-2}^{(i)}\}}{(-1)[\mu+(n-k)\lambda]\Psi_{k-1}-\mu(n-k+1)\lambda\Psi_{k-2}}$$

其中,$\Psi_0=n\lambda$, $\Phi_i^{(i)}=\lambda^i\prod_{j=0}^{i-1}(n-j)(k\gg i+1)$。

设系统有 N_g 个单组,则系统的 $MTTF_{RSRAID}$ 为:

$$MTTF_{RSRAID}=\sum_{j=0}^{k}\frac{(-1)^{j-1}\{[\mu+(n-k)\lambda]\Phi_{k-1}^{(i)}-\mu(n-k+1)\lambda\Phi_{k-2}^{(i)}\}}{N_g\{(-1)[\mu+(n-k)\lambda]\Psi_{k-1}-\mu(n-k+1)\lambda\Psi_{k-2}\}}$$

仅以 $k=2$ 的时候来说明多冗余对系统的可靠性的提高:已知 $MTBF_{disk}=800000$h, $N_g=1$, $MTTR_{disk}=2$h,计算得到 $MTBF_{RSRAID}=5333333333333333$h,由此可见 RSRAID 对系统

可靠性的提高是很可观的。

4 系统性能及可靠性测试

我们搭建了一个试验环境来测试系统的性能及可靠性，试验是在一台 1U 的服务器上进行的，具体配置如下：CPU，PIV 3.0GHz；内存，1GB；硬盘，80GB×6；操作系统使用 Fedora Core 4，内核版本为2.6.9。首先使用 IOZone 对 RSRAID 的读写性能进行了测试，图 5 和图 6 分别给出了顺序读和随机写的测试结果。

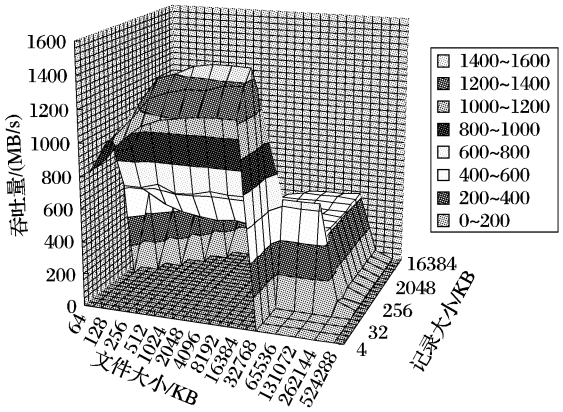

图 5 RSRAID 顺序读性能

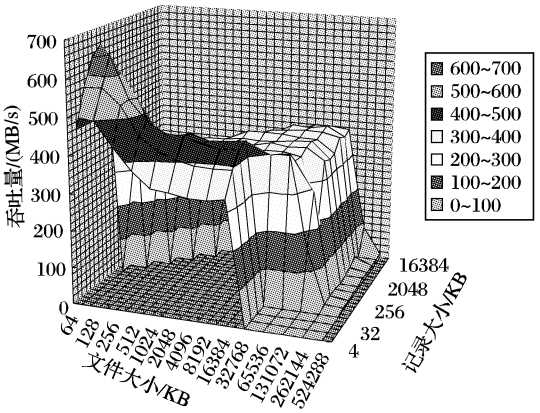

图 6 RSRAID 随机写性能

根据测试结果可知 RSRAID 具有很好的 I/O 性能，其读取数据的吞吐量最高可达到1500MB/s，其写入数据的吞吐量最高可达到500MB/s。

接着我们对系统在遭受破坏后的恢复速度进行了测试，并与 RAID6 进行了对比。

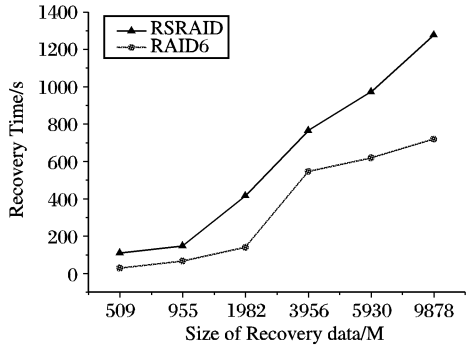

图 7 恢复时间对比测试

从图 7 中可以看到，RSRAID 的恢复时间要比 RAID6 长，这是因为虽然对 Reed-Solomon 算法进行了改进，但是与 RAID6 不同的是，每次进行解码的时候都要进行矩阵的反转运算和伽罗瓦域的乘法运算，而 RAID6 只需要 XOR 运算，时间复杂性更低。

为了测试系统的可靠性，我们分别针对阵列中有 10% ~ 50% 的磁盘损失时的可靠性进行了测试，当有 10% 的存储节点被毁坏时，数据存储的可靠性大于 99.999 999 87%，如图 8 所示。当 50% 的磁盘损坏时，数据存储的可靠性为 57% 左右，如图 9 所示。而现有的 RAID 机制，如 RAID6，只要系统中超过两个磁盘损坏，那么数据将不可恢复。可见 RSRAID 虽然在恢复速度上不如 RAID6，但是其可靠性远远优于 RAID6。

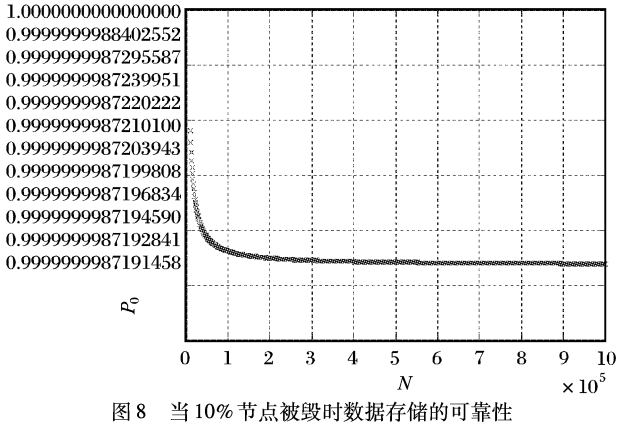

图 8 当 10% 节点被毁时数据存储的可靠性

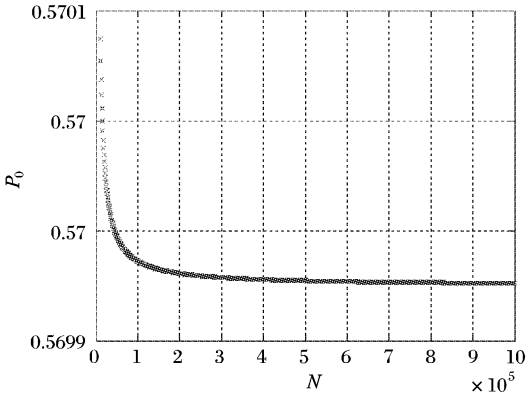

图 9 当 50% 节点被毁时数据存储的可靠性

5 结语

本文将 Reed-Solomon 算法融入到 RAID 机制中，并在 Linux 系统下实现。通过初始化时的配置，可以将系统可容忍的磁盘损坏数目增加到 $M(M \geqslant 1)$ 个，从而大大增强了系统的可靠性。通过试验验证，本系统具有良好的 I/O 性能和数据恢复性能。系统所有的编解码都由 CPU 计算完成，不需要任何特殊硬件，可以在低成本下取得高可靠性的系统。

参考文献：

[1] PATTERSON DA, GIBSON G, KATZ RH. A case for redundant arrays of inexpensive disks(RAID)[A]. 1988 ACM Conference on Management of Data[C], 1998.109 - 116.

[2] WICKER SB, BHARGAVA VK. Reed - Solomon Codes and Their Applications[M]. New York: IEEE Press, 1994.

[3] PLANK JS. A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems[J]. Software Practice & Experience, 1997, 27(9):995 - 1012.

[4] LOVE R. Linux Kernel Development[M]. Sams Publishing, 2004. 173 - 180.

[5] BURKHARD WA, MENON J. Disk array storage system reliability [A]. 23rd International Symposium on Fault-Tolerant Computing [C]. Toulouse, France, 1993.432 - 441.