Linux 中软件 RAID 的 Bitmap 功能实现分析

杨民峰　朱元忠　刘胜厚　谢丽杲　田晓玲
(北京工业职业技术学院,北京 100042)

摘　要:通过对 Linux 内核 2.6.18 版本 MD 驱动程序 bitmap 功能的实现细节的分析,探究了其实现机制,并以 Raid5 为例说明了该功能的调用关系,使读者对整个 bitmap 功能有一个全面而细致的了解,在此基础上进行了该功能的优缺点分析与评价。

关键词:Linux 内核;驱动程序;软件 Raid;Bitmap
中图分类号:TP334　　**文献标识码**:A　　**文章编号**:1671－6558(2007)02－22－05

Analysis of Bitmap Function of Software RAID in Linux

Yang Minfeng　Zhu Yuanzhong　Liu Shenghou　Xie Ligao　Tian Xiaoling
(Beijing Polytechnic College, Beijing 100042, China)

Abstract: In this paper, through the analysis of implantation detail of Bitmap function of MD drivers based on linux kernel version 2.6.18, we show the whole mechanism of this function, demonstrate the call relationship based on Raid5 codes and make the readers have a full understanding of its function. In the end we evaluate this function through the advantage and disadvantage.

Key words: Linux kernel; Devices driver; Software Raid; Bitmap

RAID是冗余磁盘阵列(Redundant Array of Inexpensive Disk)的简称。它是把多个磁盘组成一个阵列,当作单一磁盘使用。它将数据以分段(striping)的方式分散存储在不同的磁盘中,通过多个磁盘的同时读写,来减少数据的存取时间,并且可以利用不同的技术实现数据的冗余,即使有一个磁盘损坏,也可以从其他的磁盘中恢复所有的数据。

从 2.4 版内核开始 Linux 就实现了对软件 RAID 的支持,这让我们可以不必购买昂贵的硬件 RAID 设备,就能享受到增强的磁盘 I/O 性能和可靠性,进一步降低了系统的总体拥有成本。

Linux 内核 MD(MultiDevice)驱动程序实现了软件 RAID 的功能,随着 Linux 内核版本的不断推出,其软件 RAID 的功能与实现机制也不断地完善。

在 Linux 内核 2.6.18 版本中软件 RAID 增加了 bitmap 的功能,用来记录条带信息的完整性。我们对其实现细节进行详尽分析,以便对该功能有一个全面细致的了解。

1 Bitmap 功能点介绍

为了减少 RAID5 的重复同步操作数量,提高同步速度,在 2.6.18 版本 Linux 内核中实现了 bitmap 机制,记录当前设备数据同步的状况。

系统将设备划分成 chunk 大小的物理块,对于每一个块,内存中对应一个 16 位大小的元数据位图

收稿日期:2007－03－15
作者简介:杨民峰(1978－),男,山东科技大学信息学院计算机科学与技术专业毕业,助理讲师。

信息,该 16 位数据格式如图 1 所示。

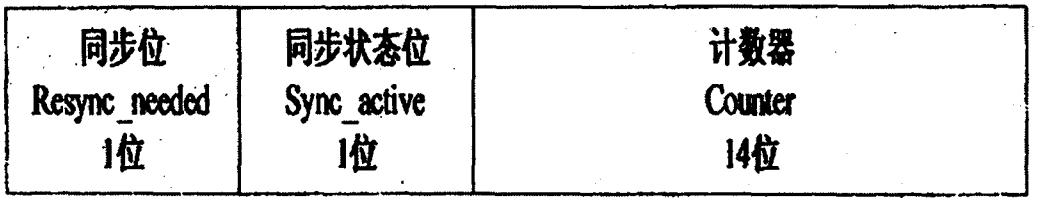

图 1　bitmap 数据结构

同步位(resync needed)在以下情况被设置:

(1)系统初始化时,从存储设备中读取位图时将该位置成 1。

(2)当某个设备上的写请求失败时该位置为 1。

(3)当正在同步的块(同步状态处于 active)其同步过程取消后将该位值设置成为 1。

同步状态位在以下情况被设置:

所有设备正在进行重新同步的操作,该位被设置为 1。

当进行写操作的时候,系统会将所对应的 bitmap 位置成 1,并且刷新到磁盘中去,只有当磁盘中的 bitmap 刷新成功之后,才开始一个正常的数据 IO 写操作。当进行 sync 操作的时候,系统会检查 bitmap 位,如果发现数据已经同步,那么跳过该数据段,不进行同步操作。采用 bitmap 策略最大的优点是减少了无谓的同步操作,提高了同步操作效率。

2 Bitmap.c 功能小结

Bitmap 的操作函数存放于 bitmap.c 文件中。纵观 bitmap.c 中的操作函数,主要完成如下几个方面的功能:

(1) 读取磁盘中 bitmap 的数据,然后在内存中建立一个 bitmap 的映射表。映射表建立成功之后,可以直接在内存中对 bitmap 进行处理,然后再刷新到磁盘中去。

(2) 一系列 bitmap 读写、置位、清零等。基本操作函数,完成 bitmap 的最基础的操作。

(3) 后台进程 bitmap-daemon-work()。该后台进程主要完成 bitmap 磁盘数据的清零操作。

(4)Bitmap 的重要进程 bitmap-unplug(),实现 bitmap 磁盘数据的置位操作。

在 RAID5.C 文件中,当进行一个写操作以及进行同步操作的时候,开始调用 bitmap 涉及的相关函数。

3 Bitmap 操作涉及的函数及说明

与 bitmap 操作相关的函数同样在 bitmap.c 文件中。函数列表如下:

3.1 bitmap-startwrite()函数说明

该函数在 add-stripe-bio()中调用。当开始一次写操作时,在 stripe 上还没有挂接任何的 bio 时,调用 bitmap-startwrite()。其调用条件是:conf—>mddev—>bitmap == 1&& firstwrite == 1。

通过 bitmap-get-counter()函数得到 bitmap counter。然后通过 bitmap-file-set-bit()函数设置相应的 bitmap 位,在 bitmap-file-set-bit()函数中,设置 BITMAP-PAGE-DIRTY 标记,那么在 raid5d()函数中会调度 bitmap-unplug()函数对 bitmap 的 super-block 进行刷新。

3.2 bitmap-unplug()函数说明

该函数在 raid5d()守护进程中调度。

当进行一次写操作的时候,需要将内存中映射的 bitmap 相应位置成 1,并且需要将内存 bitmap 信息刷新到磁盘上去,这样的话,在正常写数据的过程中如果发生错误、故障,那么条带信息还能保持同步。由于 bitmap 的置位操作和清零操作权限不一致,置位操作具有很高权限,而清零操作的权限较低,因此分别采用两个函数分别实现读操作过程中的置位和清零操作。Bitmap-unplug()函数就是完成了权限较高的置位操作。

函数的实现过程比较简单,通过判断 BITMAP-PAGE-DIRTY 标记确认某个 bitmap page 中的位是否被设置了(dirty)? 是否需要进行一次数据写操作? 另外,还需要判断 BITMAP-PAGE-NEEDWRITE 标记是否有效,如果 BITMAP-PAGE-DIRTY 标记或者 BITMAP-PAGE-NEEDWRITE 标记有效,那么可以调用 write-page()进行一次 bitmap 写操作。

3.3 bitmap-start-sync()函数说明

该函数在 sync-request()函数中调用。

在 sync-request()函数中,如果驱动程序采用了 bitmap 的策略,那么需要检测 bitmap 中对应的位,如果相应的数据块需要同步,那么,会继续执行同步

操作,如果已经同步,那么 bitmap-start-sync()会返回0,并且 blocks 数据域中的数据将会有效,sync-request()函数会直接返回不需要同步的数据量。这就做到了同步操作的数据过滤,避免将已经同步的数据再次同步。

该函数通过 bitmap-get-counter()函数获取 bmc 指针。当磁盘阵列不处于 degraded 状态时,通过如下语句设置 bitmap 同步操作标记位:

```
* bmc | = RESYNC-MASK
```

3.4 bitmap-daemon-work()函数说明

这是 bitmap 清零操作的重要进程,通过多次调用 bitmap-daemon-work()函数实现 bitmap 相应位置的清零操作。在 bitmap 的操作过程中,由于 bitmap 置1操作很重要、很谨慎,需要在磁盘写数据之前将 bitmap superblock 相应位置1,因此,这个功能由 bitmap-unplug()函数来单独完成。Bitmap 的清零操作权限较低,不构成数据的危害性,因此可以延迟,分多个阶段将数据写入磁盘,这个功能就由 bitmap-daemon-work()函数单独完成。

该函数由 md-check-recovery()函数调用,由于刚进 md-check-recovery()函数就调用 bitmap-daemon-work(),所以,bitmap-daemon-work()就是由 raid5d()守护线程调用的。

函数主要功能是周期性的被唤醒,去清除标记位并且将新数据不断刷新到磁盘阵列中去。

该函数涉及的重要标记:

1、BITMAP-PAGE-CLEAN:需要将内存中对应的 bitmap 相应位清零。

2、BITMAP-PAGE-NEEDWRITE:需要将内存中的 bitmap 数据写入磁盘。

通过 bmc 计数器和上述两个重要标记,可以将 bitmap-daemon-work()函数分成如下5个功能:

(1) 当 BITMAP-PAGE-CLEAN 无效,BITMAP-PAGE-NEEDWRITE 有效时,进行磁盘数据刷新操作。

(2) 当 BITMAP-PAGE-CLEAN 有效,BITMAP-PAGE-NEEDWRITE 有效,并且 * bmc = = 0 时,调用 write-page()函数磁盘上的 bitmap 区域。

(3) 当 BITMAP-PAGE-CLEAN 有效,BITMAP-PAGE-NEEDWRITE 无效,并且 * bmc = = 0 时,清除 BITMAP-PAGE-CLEAN 标记,设置 BITMAP-PAGE-NEEDWRITE 标记有效。

(4) 当 BITMAP-PAGE-CLEAN 有效,BITMAP-PAGE-NEEDWRITE 无效,并且 * bmc = = 2 时,将 * bmc - - 。

(5) 当 BITMAP-PAGE-CLEAN 有效,BITMAP-PAGE-NEEDWRITE 无效,并且 * bmc = = 1 时,将 * bmc - - ,并且将内存中的 bitmap 相应位清零。

该函数的操作过程如下:

在 BITMAP-PAGE-CLEAN 无效,即已经将内存中的 bitmap 相应位清零时,判断 BITMAP-PAGE-NEEDWRITE 刷新磁盘标记是否有效,如果无效,那么将设置该标记位有效。如果有效,那么直接调用 write-page()函数将内存中的 bitmap 刷新到对应的磁盘中去。当一个 bitmap page 处理完毕之后,采用 continue 指令进行 for 循环。

当 bitmap page 中的 BITMAP-PAGE-CLEAN 有效时,调用 bitmap-get-counter()函数获得该数据块对应的 bmc,如果 * bmc = = 2 时,那么将 * bmc - - ,在下一次进入的时候再进行内存 bitmap 的清零操作,从这一点上可以看出 bitmap 的清零操作优先级是比较低的。如果 * bmc = = 1,那么 * bmc - - ,同时将内存中的 bitmap 相应位清零。

当 bitmap page 中的 BITMAP-PAGE-CLEAN 有效时,并且内存中的 bitmap 相应位已经清零之后,进入如下结构的代码区:

```
If ( ! lastpage ! = NULL ) {
    ....
}
```

在该代码区首先检查 BITMAP-PAGE-NEEDWRITE 是否有效,如果该标记位有效,那么调用 write-page()函数将相应的 page 写到磁盘中去。如果 BITMAP-PAGE-NEEDWRITE 标记位无效时,那么设置该标记位有效,等到下一次调度 bitmap-daemon-work()函数时再将相应 page 写入磁盘。

在正常操作过程中,需要清除 Bitmap 标记,并且将标记刷新到磁盘中去。其主要经历如下几个阶段:

(1) Bitmap-endwrite ( ), BITMAP-PAGE-CLEAN 标记有效。

(2)第一次调度 bitmap-daemon-work(), * bmc - - ,BITMAP-PAGE-CLEAN 标记有效。

(3)第二次调度 bitmap-daemon-work(), * bmc - - ,清除内存映射的 bitmap 相关标记,清除 BITMAP-PAGE-CLEAN 标记,一次循环以后,设置 BITMAP-PAGE-NEEDWRITE 标记有效。

(4)第三次调度 bitmap-daemon-work(),调用

write-page()将对应 bitmap 刷新到磁盘中去。

从上述操作过程，我们也可以看出 bitmap 的清零操作权限是如此之低(竟然还用 bmc 计数器进行操作等待)，完成一次磁盘 bitmap 信息的清除需要三次调度 bitmap-daemon-work()函数，其参考脚本还可以在 bitmap-flush()函数中找到。在 bitmap-flush()函数中三次调用了 bitmap-daemon-work()函数，实际上就是完成了一个完整的数据清零过程，少调用一次都不行。

3.5 其他涉及函数说明

1、bitmap-endwrite()：在 handle-stripe()函数中调用，当完成写操作的时候，调用此函数，设置 BITMAP-PAGE-CLEAN 标记，然后通过 bitmap-daemon-work()函数清除 bitmap 的相应位。

2、bitmap-write-all ()：设置 BITMAP-PAGE-NEEDWRITE 标记，由后台进程具体执行写操作。

3、bitmap-create()：初始化一个 bitmap 的内存结构，如果初始化失败，那么调用 bitmap-destroy()函数释放掉已经分配的内存。在 bitmap-create()函数中，需要调用 bitmap-init-from-disk()函数将磁盘中的 bitmap 信息读入内存，并且在内存建立 filemap，与磁盘中的 bitmap 一一对应。

4、bitmap-destroy()：将已经分配的内存空间释放掉，真正的执行函数是 bitmap-free()。

5、bitmap-update-sb()：该函数实现超级块(superblock)的更新，并且将该超级块更新到磁盘中去。

6、bitmap-get-counter()：得到 page 所对应的 bmc 计数器，需要注意的是这段程序代码中的 block 数据域，这个数据域返回已经同步的数据量，在 md-do-sync()函数中将用到 block 数据域的值。

7、bitmap-file-set-bit()：设置内存映射 bitmap 对应位，并且使能 BITMAP-PAGE-DIRTY 标记，表明需要将内存映射的 bitmap 数据刷新到磁盘中去。

8、bitmap-checkpage()：该函数用于为 bitmap page 分配 counter，如果已经分配了 counter，那么返回 0，如果没有分配，那么调用 bitmap-alloc-page()函数为其一个 page，挂接到 bitmap—>bp[page].map 上。如果分配 page 失败，那么置位 hijacked，并且返回 0。

9、write-page()：这个函数实现了磁盘数据刷新功能，其采用了两种数据刷新方法，一种方法是当 bitmap 结构体中存在 file 指针时，会构造一个 buffer，然后通过 submit-bh()发送数据请求。另一种方法是当 bitmap 结构体中不存在 file 指针时，会调用 write-sb-page()函数，采用 bio 请求方法将数据发送下去。

4 Bitmap 的可靠刷新机制

当进行每次写操作的时候，都需要将 bitmap 刷新一次(如果不进行可靠的刷新，那么有可能导致磁盘条带数据的不同步)。Bitmap 的刷新是指将内存中的 bitmap 数据正确的刷新到磁盘中去。这就涉及到 bitmap 的可靠刷新机制。在这里面所谓的“bitmap 可靠刷新机制”是指 bitmap 相应位置 1 之后，需要可靠的刷新到磁盘中去，并且需要在正常数据写操作之前完成 bitmap 数据的磁盘刷新。

在 MD 驱动中，bitmap 的可靠刷新实现的很微妙，bitmap 在内存中置位以后，需要将 bitmap 数据刷新到磁盘中去，为了保证条带信息的同步，bitmap 信息必须在数据写操作之前刷新完毕，因此，系统在 add-stripe-bio()的函数中调用了 bitmap-startwrite()函数进行 bitmap 刷新操作。Bitmap-startwrite()函数本身并没有调用 write-page()函数执行写操作，而是调用了 bitmap-file-set-bit()函数，设置了内存映射的 bitmap，然后设置了 BITMAP-PAGE-DIRTY 标记。

不在 bitmap-startwrite()函数中直接调用 write-page()函数实现 bitmap 的磁盘刷新操作的主要原因在于块设备的 IO 读写操作是通过 queue 队列进行的，不能保证每次 IO 写操作能够及时地完成，因此如果直接进行写操作的话，可能会存在 bitmap 刷新和数据写操作次序之间的颠倒。

真正处理 BITMAP-PAGE-DIRTY 标记的函数是 bitmap-unplug()，该函数在 raid5d()守护进程中调用，其调用条件需要满足：Seq-flush > seq-write。在 add-stripe-bio()函数调用 bitmap-startwrite()函数之后，通过 STRIPE-BIT-DELAY 、mddev—>md-seq 和 seq-write 三个标记将 sh 操作队列挂接到 bitmap-list 下面，实现数据写操作的延迟，等待调用 bitmap-unplug()函数刷新 bitmap 磁盘数据。另一方面当系统调用 raid5-unplug-device()函数之后，会将 seq-flush++，进入 raid5d()守护进程之后，满足 seq-flush > seq-write 的条件，然后调用 bitmap-unplug()函数，实现磁盘 bitmap 数据的刷新。当磁盘 bitmap 数据刷新完毕之后，调用 active-bit-delay()及 release-stripe()函数，将 bitmap-list 下面的 sh 队列挂接到 handle-list 下面去，实现 sh 数据的正常写操作。

综上所过可以看出，bitmap 的磁盘刷新必须在数据写操作之前完成。状态流图如图 2 所示。

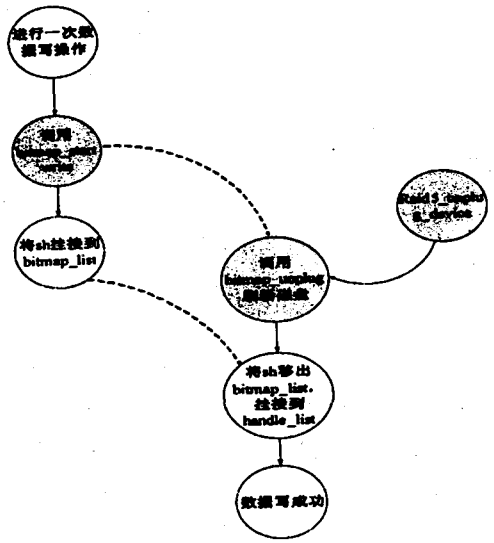

图2 bitmap的刷新机制状态流图

5 优缺点分析

2.6.12版本同步操作和recovery操作是相差不多的，但是在2.6.18版本中，同步操作采用了bitmap策略，大大提高了同步操作的时间。

在md-do-sync函数中，如果MD-RECOVERY-SYNC标记有效，说明需要进行一个同步操作，那么会进行如下条件的判断：

1、检查bitmap指针是否有效，即bitmap是否存在?

2、看MD-RECOVERY-REQUESTED是否有效，无效则不进行recovery操作。

如果bitmap不存在，MD-RECOVERY-REQUESTED标记无效，那么采用2.6.12提供的策略，即check point策略，将mddev—>recovery-cp作为同步操作的起始位置。

在md-do-sync()函数中，循环调用sync-request()函数，进行同步/恢复操作。

在sync-request()函数中，如果同步操作采用bitmap策略，那么需要对bitmap进行检查，调用bitmap-start-sync()函数检查位图信息，如果对应的扇区需要做同步操作，那么设置sync-active标记，并且返回1，如果对应扇区正在做同步操作，同样返回1，无需做同步操作，返回0。

如果对应扇区无需做同步操作，返回0，并且从sync-blocks数据域得到具体同步块的大小。由于每次IO数据操作是以STRIPE-SECTOR为基本元素的，所以需要对sync-blocks做相应的取整操作，关键代码如下：

```
Sync-blocks /= STRIPE-SECTORS;
Return sync-blocks * STRIPE-SECTORS;
```

从上面的分析可以看到，采用bitmap策略减少了无谓的数据同步操作，大大减轻了数据IO负载，提升了软件RAID的性能。

但是在写操作的过程中，每次都需要保证bitmap刷新一遍，并且延迟数据写操作，会对写操作造成不良影响。

6 结语

通过对Linux内核2.6.18的源程序的阅读分析，整理了MD驱动程序中bitmap的功能实现细节及使用说明，并总结其优缺点，希望对大家的相关学习起到抛砖引玉的作用。

参考文献：

[1] Derek Vadala. Linux Raid管理(影印版)[M]. 北京：清华大学出版社，2003

[2]RobertLove. Linux内核设计与实现[M]. 陈莉君译. 北京：机械工业出版社，2004

[3]Daniel P Bovet. 深入理解Linux内核[M]，陈莉君译. 北京：中国电力出版社，2004

[4]David Mosberger, Stephane Eranian. IA—64 Linux内核设计与实现[M]. 梁金昆译. 北京：清华大学出版社，2004

[5]Klaus Wehrle. Linux网络体系结构—Linux内核中网络协议的设计与实现[M]. 汪青青译. 北京：清华大学出版社，2006

[6]赵炯. Linux内核完全注释[M]，北京：机械工业出版社，2005

(责任编辑：张艳霞)