Linux 系统中 SLAB 分配器内存回收算法分析和优化

陈卉[1]，张振华[2]
[1]北京邮电大学通信与信息工程学院，北京 (100044)
[2]北京邮电大学计算机科学与技术学院，北京 (100876)
E-mail: ch19860304@gmail.com

摘　要：SLAB 分配器在 linux 系统中的使用大大提高了内核中分配内存对象的效率，并且解决了小对象分配内存时产生的内存碎片问题。而该分配器的“内存回收算法”更是保证了其内核空间的充分利用。本文首先简单地介绍了 slab 分配器的总体结构及其与“内存回收算法”息息相关的关键数据结构，然后详细地描述了“内存回收算法”的具体机制，最后对该算法提出了优化策略并给出了其统计数据及仿真结果。

关键词：SLAB 分配器；内存回收；优化策略
中图分类号：TP316.81

1．引言

“内存回收”，又名“内存垃圾回收”，是操作系统内存管理中最重要的组成部分之一，而 SLAB 分配器管理的是 linux 系统中宝贵的内核空间，因此其中的“内存回收”对内核空间的充分利用起着举足轻重的作用，但目前并没有介绍有关 SLAB 分配器“内存回收”的学术论文。故本文在就 linux 2.26.25 部分源代码进行细致分析后，对 SLAB 分配器的“内存回收”进行了详细的分析与优化工作。

2．SLAB 分配器的关键数据结构

SLAB 分配器[4]的总体结构如图 1 所示，主要分为三层结构：高速缓存、slab、对象。其中，高速缓存是用来缓存某种特定内核变量的缓冲区；slab 则是高速缓存中的一个个大小固定的内存分区；对象是 slab 中的一个个大小固定的小内存块，是存储变量的单位。SLAB 分配器通过将已经释放的对象缓存在高速缓存区中，而不是真正地释放回内存中，大大提高了 LINUX 系统内核空间中“分配对象”的效率。

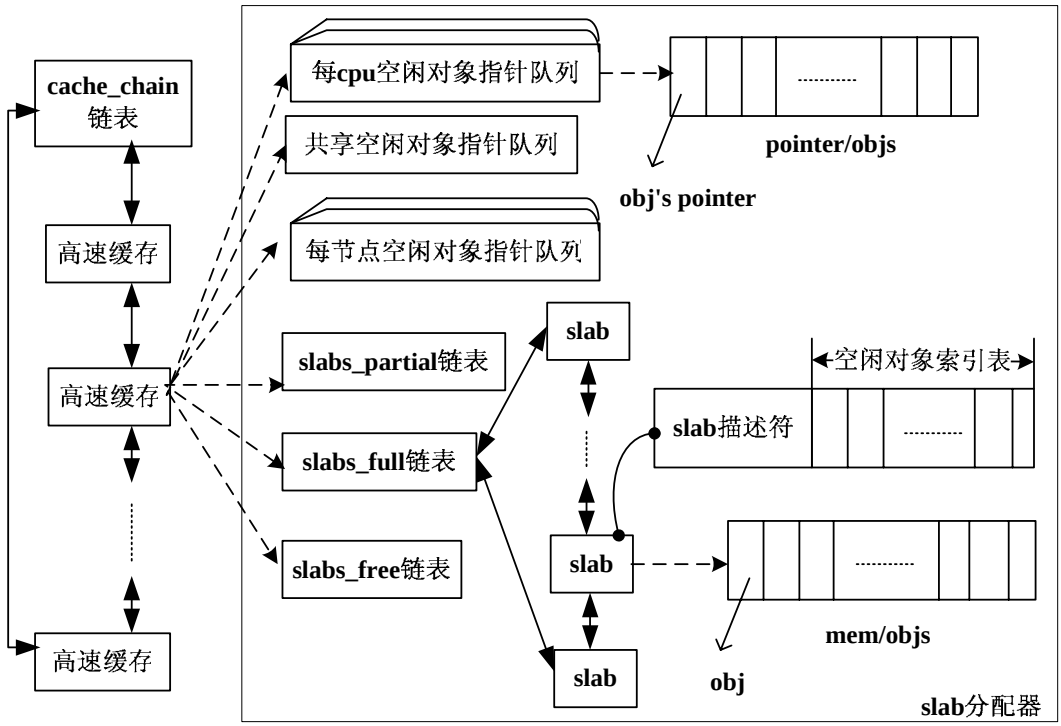

图 1　SLAB 分配器总体结构图

2.1　l 3 链表

如图 1 所示，每个高速缓存包含 slabs_free，slabs_partial，slabs_full 三个链表，本文将其称为“l3 链表”[2]。每个链表的后面都链接着多个 slab，并标志着其属下 slab 分区的状态。当某 slab 分区处于部分空闲状态（部分对象在使用，部分对象未使用）时，其将会被加入到 slabs_partial 链表中；当某 slab 分区处于全满状态（所有对象全部被使用）时，其将会被加入到 slabs_full 链表中；当某 slab 分区处于空闲状态（所有对象全部未使用）时，其将会被加入到 slabs_free 链表中。

显然，一个新的对象申请到达时，slab_partial 链表中首个 slab 分区的对象会被优先考虑，只有当 slab_partial 链表中不存在任何 slab 分区时，slabs_free 链表中的 slab 分区才会被考虑；而一个对象使用完毕释放时，它将被释放回其原属 slab 中。在申请或释放的过程中，如果一个 slab 分区的状态（空闲状态、部分空闲状态、全满状态）发生改变的话，必须将其从原先的链表转移到新的链表中。

由于该 slabs_free 链表中的所有 slab 分区均处于空闲状态，“内存回收算法”可以方便从该链表中销毁其中的 slab 分区来节省内存。

2.2 空闲对象指针队列

每当从 slab 分区中取出或还回一个对象时，会有两项动作：第一，更新空闲对象索引表［注 1］。第二，如果该 slab 分区的状态发生了变化，还得将其从一个链表队列转移到另一个链表队列。显然这是非常费劲的。

为了提高 slab 分配器取还对象的效率，SLAB 分配器提出了一个概念：“空闲对象指针队列”[1]，通过一次性的从多个 slab 分区中取出大批对象，此时完成更新空闲对象索引表、转移队列的工作，并将这些对象的指针存放在“空闲对象指针队列”中。通过该方法在以后每

次取或还对象时，都可以直接操作“空闲对象指针队列”，从而大大提高效率。

“空闲对象指针队列”包含三个部分：“每 cpu 空闲对象指针队列”、“共享空闲对象指针队列”、“每节点空闲对象指针队列”。它们在取对象或还对象的时候扮演着重要角色，而这方面由于不是本文的重点，不作介绍。这三部分能存放指针的最大容量如下表所示，它们与对象的大小(size)有关，其中，ac->limit 表示“每 cpu[1]空闲对象指针队列”的容量、share->limit 表示“共享空闲对象指针队列” 的容量、alien->limit 表示“每节点空闲对象指针队列” 的容量。

表 1 “空闲对象指针队列”容量

对象的大小范围	ac->limit	share->limit	alien->limit
size<=256	120	480	12
256<size<=1024	54	216	12
1024<size<=4096	24	0	12
4096<size<=131072	8	0	12
131072<size	1	0	1

从表 1 可以看出， 对某些高速缓存来说，三个“空闲对象指针队列”，特别是“每 cpu 空闲对象指针队列”和“共享空闲对象指针队列” 容量巨大，如果该高速缓存使用不太频繁，但“空闲对象指针队列”中又存放大量空闲对象指针，这会是一笔巨大的内存浪费。SLAB 分配器的“内存回收算法”会对这一情况进行处理。

［注 1］每个 slab 分区都对应一张“空闲对象索引表”，通过该表，可用来查找该 slab 分区中空闲对象。取对象时，从“空闲对象索引表”中检索出当前空闲对象，然后更新该表，以标志该对象已被使用。释放对象时，也要更新该表，以标志该对象空闲。

3. 内存回收算法

3.1 算法介绍

随着 SLAB 机制[5]在内核中的使用，频繁的对象申请会使得高速缓存中的 slab 的数目越来越多，缓存区的空间会越来越大。而还回去的那部分对象也只是还到在其所属的“slab 分区”中，并没有被真正的“释放”，故内存将会被耗尽。因此 SLAB 分配器需要合适的内存回收算法来回收高速缓存中部分无用的“slab 分区”，将其真正地释放。

SLAB 分配将“内存回收”作内核中的一个工作任务，挂接着内核预定义的事件工作队器列——调度器（scheduler）队列 keventd_wq[3]，由内核中专门的内核线程 eventd 去完成这个队列中的所有工作任务。内核线程 eventd 首先循环遍历 cache_chain 链表中的所有高速缓存，并对满足“回收条件”的各个高速缓存进行空闲页框回收。每完成一次这样的“内存回收”，或在试图完成“内存回收”时获取内核锁失败时，就延迟 2s 后，再将该“内存回收”工作挂接到调度工作队列 keventd_wq 中再次被调度。其中“回收条件”有两个：

1、该高速缓存已经达到了“回收时间 reap_time”。该值为 4 秒，即该高速缓存上次的“内存回收”与本次“内存回收”至少要事隔 4s。

2、该高速缓存中的“免回收标志位 free_touch”没有置位。当某高速缓存中的空闲对象不够用时，其“免回收标志位 free_touch”会被置位，此时，不需要对该高速缓存进行“内存回收”，否则就要进行内存回收。

“内存回收”算法流程图[6]如图 2 所示，具体描述如下：

1、清空“每节点空闲对象指针队列”中所有空闲对象指针，清理“本地空闲对象指针队列”和“共享空闲对象指针队列”中 tofree 个空闲对象指针；将这些指针对应的对象释放回其所属 slab 中，这是为了保证尽可能多产生全空闲状态的 slab 分区，从而达到最大程度的内存回收。其中，tofree 的值是根据经验公式所得：

$$tofree = \begin{cases} (\lim it + 4)/5 & avail \geq (\lim it + 4)/5 \\ (avail + 1)/2 & avail < (\lim it + 4)/5 \end{cases} \quad (1)$$

公式 1 中，avail 为目前“本地空闲对象指针队列”或“共享空闲对象指针队列”中空闲对象指针的个数，而 limit 为其容量（见表 1）。

2、从 slabs_free 队列中销毁 nr_free 个 slab 分区，将其占用的页还回到内存中，nr_free 值是根据经验公式所得，但最大只能是 slabs_free 队列中 slab 的个数：

$$nr\_free = \frac{(1 + nrcpu) * batchcount + 6 * objnr - 1}{5 * objnr} \quad (2)$$

公式 2 中，Objnr 为每个 slab 中对象的个数， nrcpu 为系统中 cpu 的个数，batchcount 的值为表 1 中 ac->limit 的值加 1 后除 2 的结果。

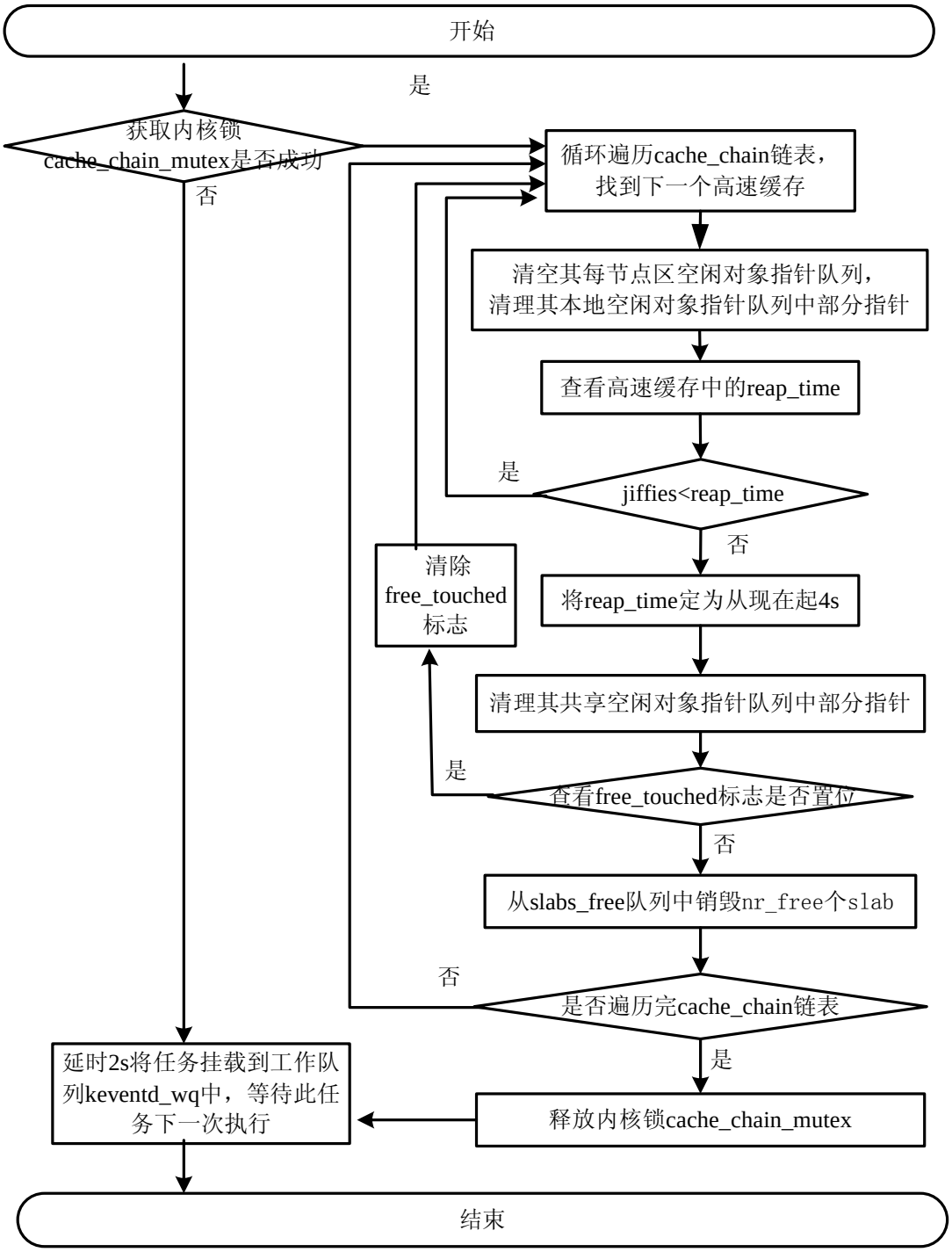

图 2 内存回收算法流程图

3.2 算法优化

从 3.1 节可以看出，SLAB 分配器的内存回收策略，过于单调统一。它仅仅把高速缓存分为两种：“内存紧缺型”、非内存紧缺型。如果某高速缓存的“免回收标志位 free_touch”置位了，即为“内存紧缺型”，对其采取不回收的策略。而对于“非内存紧缺型”高速缓存，仅回收其中部分内存。而当系统中“非内存紧缺型”高速缓存大部分处于“极少使用”的状态时，仅回收其部分内存是不够的。

本文采取的优化策略，就是基于这一点考虑，将“非内存紧缺型”的高速缓存进一步细分为“正常使用型”与“不常使用型”，而对于不同类型的高速缓存采用不同的内存回收策略，以而改善其性能。

如果某高速缓存连续在“回收时间 reap_time”内仍没有被使用（分配对象），其会被标志为“不常使用型”。反之，则为“正常使用型”。对于“不常使用型”高速缓存，回收它的内存时，清空所有“空闲对象指针队列”中的所有空闲对象指针，销毁 slabs_free 队列中所有 slab 分区；对于“正常使用型”的高速缓存来说，采取原 slab 内存回收策略（3.1 节）。

仿真与统计结果显示，在单 cpu（非 SMP）系统下，理想情况下该优化算法在空间上仅有 0.73%左右的内存节省，且会带来性能的下降；而在多 cpu（SMP）系统下，该优化算法在空间性能的提升会随着 cpu 的个数成倍增加，如，当某系统的 cpu 个数达到十个时，其性能的提升为 7.3%，而且此时性能下降的影响也会降低。故此优化算法适合于 SMP 架构的系统。

4. 总结

本文在文章开头简要概括了 SLAB 分配器的结构和“内存回收算法”的重要性，然后重点介绍了 slab 机制中的关键数据结构和内存回收算法。并针对 slab 内存回收机制过于单一的缺陷进行了改进，对内存回收的时机进行了更加详细的描述，使之更加适合在复杂环境下的应用。

参考文献

[1] Daniel P. Bovet, Marco Cesa 著. 深入理解 Linux 内核（第三版）[M], 北京：中国电力出版社.2004 年
[2] Michael Beck 等. Linux 内核编程指南[M]. 北京:清华大学出版社.2005
[3] Robert Love 等. Linux 内核设计与实现[M]. 北京: 机械工业出版社.2005
[4] 洪津津.石教英. Linux 中的 Slab 分配器[J]. 计算机应用研究.2006.5. 第 11 期:82-84
[5] 赵鲲鹏 , 苏葆光. Linux 内存管理中的 Slab 分配机制[J]. 现代计算机.2000.3.7. 第 235 期:89-91
[6] Linux 2.6.25 源代码

Analysis and Optimize of SLAB Allocator’s Memory Reap Algorithm under linux

Chen hui[1]， Zhang zhenhua[2]
1 School of Communication and Information Engineering ,Beijing university of Post and Telecommunity, Beijing (100044)
2 School of Computer Science and Technology, Beijing University of Post and Telecommunity, Beijing(100876)

Abstract
SLAB Allocator which is used on linux system improve memory's efficiency greatly, and avoiding internal fragmentation when allocating for small memory object. Its memory reap algorithm make the better use of kernel areas. First of all, this paper introduce SLAB Allocator's main and key structure which is relevant to "memory reap algorithm"; second, it represent the mechanism of "memory reap algorithm"; finally, this paper propose a strategy of optimizing about it, and give its statistical data and simulation results.
Keywords: The Slab Allocator;memory reap;strategy of optimizing

作者简介：
陈卉，女，1986 年生，硕士研究生，主要研究方向是 linux 内核中内存算法。