Subject: [PATCH 3/3][RFC] Containers: Pagecache controller reclaim
Posted by Vaidyanathan Srinivas on Wed, 21 Feb 2007 14:24:54 GMT
View Forum Message <> Reply to Message

The reclaim code is similar to RSS memory controller. Scan control is
slightly different since we are targeting different type of pages.

Additionally no mapped pages are touched when scanning for pagecache pages.

RSS memory controller and pagecache controller share common code in reclaim
and hence pagecache controller patches are dependent on RSS memory controller
patch even though the features are independently configurable at compile time.

Signed-off-by: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>

```
---
 include/linux/memctlr.h |    6 ++++
 mm/memctlr.c            |    4 +-
 mm/pagecache_acct.c     |   11 +++++++-
 mm/vmscan.c             |   65 +++++++++++++++++++++++++++++++++---------
 4 files changed, 71 insertions(+), 15 deletions(-)

--- linux-2.6.20.orig/include/linux/memctlr.h
+++ linux-2.6.20/include/linux/memctlr.h
@@ -19,6 +19,12 @@ enum {
  MEMCTLR_DONT_CHECK_LIMIT = false,
 };

+/* Type of memory to reclaim in shrink_container_memory() */
+enum {
+ RECLAIM_MAPPED_MEMORY = 1,
+ RECLAIM_PAGECACHE_MEMORY,
+};
+
 #ifdef CONFIG_CONTAINER_MEMCTLR
 #include <linux/wait.h>

--- linux-2.6.20.orig/mm/memctlr.c
+++ linux-2.6.20/mm/memctlr.c
@@ -146,8 +146,8 @@ static int memctlr_check_and_reclaim(str
     nr_pages = (pushback * limit) / 100;
    mem->reclaim_in_progress = true;
    spin_unlock(&mem->lock);
-   nr_reclaimed += memctlr_shrink_mapped_memory(nr_pages,
-        cont);
+   nr_reclaimed += shrink_container_memory(
+     RECLAIM_MAPPED_MEMORY, nr_pages, cont);
    spin_lock(&mem->lock);
    mem->reclaim_in_progress = false;
    wake_up_all(&mem->wq);
--- linux-2.6.20.orig/mm/pagecache_acct.c
+++ linux-2.6.20/mm/pagecache_acct.c
@@ -30,6 +30,7 @@
 #include <asm/div64.h>
 #include <linux/klog.h>
 #include <linux/pagecache_acct.h>
+#include <linux/memctlr.h>

 /*
  * Convert unit from pages to kilobytes
@@ -338,12 +339,20 @@ int pagecache_acct_cont_overlimit(struct
   return 0;
 }

-extern unsigned long shrink_all_pagecache_memory(unsigned long nr_pages);
+extern unsigned long shrink_container_memory(unsigned int memory_type,
+    unsigned long nr_pages, void *container);

 int pagecache_acct_shrink_used(unsigned long nr_pages)
 {
  unsigned long ret = 0;
  atomic_inc(&reclaim_count);
+
+ /* Don't call reclaim for each page above limit */
+ if (nr_pages > NR_PAGES_RECLAIM_THRESHOLD) {
+  ret += shrink_container_memory(
+    RECLAIM_PAGECACHE_MEMORY, nr_pages, NULL);
+ }
+
  return 0;
 }

--- linux-2.6.20.orig/mm/vmscan.c
+++ linux-2.6.20/mm/vmscan.c
@@ -43,6 +43,7 @@

 #include <linux/swapops.h>
 #include <linux/memctlr.h>
+#include <linux/pagecache_acct.h>

 #include "internal.h"

@@ -70,6 +71,8 @@ struct scan_control {

  void *container;  /* Used by containers for reclaiming */
      /* pages when the limit is exceeded  */
+ int reclaim_pagecache_only;     /* Set when called from
+       pagecache controller */
 };

 /*
@@ -474,6 +477,15 @@ static unsigned long shrink_page_list(st
    goto keep;

   VM_BUG_ON(PageActive(page));
+  /* Take it easy if we are doing only pagecache pages */
+  if (sc->reclaim_pagecache_only) {
+   /* Check if this is a pagecache page they are not mapped */
+   if (page_mapped(page))
+    goto keep_locked;
+   /* Check if this container has exceeded pagecache limit */
+   if (!pagecache_acct_page_overlimit(page))
+    goto keep_locked;
+  }

   sc->nr_scanned++;

@@ -522,7 +534,8 @@ static unsigned long shrink_page_list(st
   }

   if (PageDirty(page)) {
-   if (referenced)
+   /* Reclaim even referenced pagecache pages if over limit */
+   if (!pagecache_acct_page_overlimit(page) && referenced)
     goto keep_locked;
    if (!may_enter_fs)
     goto keep_locked;
@@ -849,6 +862,13 @@ force_reclaim_mapped:
   cond_resched();
   page = lru_to_page(&l_hold);
   list_del(&page->lru);
+  /* While reclaiming pagecache make it easy */
+  if (sc->reclaim_pagecache_only) {
+   if (page_mapped(page) || !pagecache_acct_page_overlimit(page)) {
+    list_add(&page->lru, &l_active);
+    continue;
+   }
+  }
   if (page_mapped(page)) {
    if (!reclaim_mapped ||
        (total_swap_pages == 0 && PageAnon(page)) ||
@@ -1044,6 +1064,7 @@ unsigned long try_to_free_pages(struct z
   .swap_cluster_max = SWAP_CLUSTER_MAX,
   .may_swap = 1,
   .swappiness = vm_swappiness,
+  .reclaim_pagecache_only = 0,
  };

  count_vm_event(ALLOCSTALL);
@@ -1148,6 +1169,7 @@ static unsigned long balance_pgdat(pg_da
   .may_swap = 1,
   .swap_cluster_max = SWAP_CLUSTER_MAX,
   .swappiness = vm_swappiness,
+  .reclaim_pagecache_only = 0,
  };
  /*
   * temp_priority is used to remember the scanning priority at which
@@ -1378,7 +1400,7 @@ void wakeup_kswapd(struct zone *zone, in
  wake_up_interruptible(&pgdat->kswapd_wait);
 }

-#if defined(CONFIG_PM) || defined(CONFIG_CONTAINER_MEMCTLR)
+#if defined(CONFIG_PM) || defined(CONFIG_CONTAINER_MEMCTLR) ||
defined(CONFIG_CONTAINER_PAGECACHE_ACCT)
 /*
  * Helper function for shrink_all_memory().  Tries to reclaim 'nr_pages' pages
  * from LRU lists system-wide, for given pass and priority, and returns the
@@ -1455,6 +1477,7 @@ unsigned long shrink_all_memory(unsigned
   .swap_cluster_max = nr_pages,
   .may_writepage = 1,
   .swappiness = vm_swappiness,
+  .reclaim_pagecache_only = 0,
  };

  current->reclaim_state = &reclaim_state;
@@ -1531,33 +1554,50 @@ out:
 }
 #endif

-#ifdef CONFIG_CONTAINER_MEMCTLR
+#if defined(CONFIG_CONTAINER_MEMCTLR) ||
defined(CONFIG_CONTAINER_PAGECACHE_ACCT)
+
 /*
  * Try to free `nr_pages' of memory, system-wide, and return the number of
  * freed pages.
  * Modelled after shrink_all_memory()
  */
-unsigned long memctlr_shrink_mapped_memory(unsigned long nr_pages, void *container)
+
+unsigned long shrink_container_memory(unsigned int memory_type, unsigned long nr_pages,
void *container)
 {
  unsigned long ret = 0;
  int pass;
  unsigned long nr_total_scanned = 0;
-
+ struct reclaim_state reclaim_state;
  struct scan_control sc = {
   .gfp_mask = GFP_KERNEL,
-  .may_swap = 0,
   .swap_cluster_max = nr_pages,
   .may_writepage = 1,
-  .swappiness = vm_swappiness,
-  .container = container,
-  .may_swap = 1,
-  .swappiness = 100,
  };

+ switch (memory_type) {
+  case RECLAIM_PAGECACHE_MEMORY:
+   sc.may_swap = 0;
+   sc.swappiness = 0; /* Do not swap, only pagecache reclaim */
+   sc.reclaim_pagecache_only = 1; /* Flag it */
+   break;
+
+  case RECLAIM_MAPPED_MEMORY:
+   sc.container = container;
+   sc.may_swap = 1;
+   sc.swappiness = 100; /* Do swap and free memory */
+   sc.reclaim_pagecache_only = 0; /* Flag it */
+   break;
+
+  default:
+   BUG();
+ }
+ current->reclaim_state = &reclaim_state;
+
  /*
   * We try to shrink LRUs in 3 passes:
   * 0 = Reclaim from inactive_list only
-  * 1 = Reclaim mapped (normal reclaim)
+  * 1 = Reclaim from active list
+  *  (Mapped or pagecache pages depending on memory type)
   * 2 = 2nd pass of type 1
   */
  for (pass = 0; pass < 3; pass++) {
@@ -1565,7 +1605,6 @@ unsigned long memctlr_shrink_mapped_memo

   for (prio = DEF_PRIORITY; prio >= 0; prio--) {
    unsigned long nr_to_scan = nr_pages - ret;
-
    sc.nr_scanned = 0;
    ret += shrink_all_zones(nr_to_scan, prio,
       pass, 1, &sc);
@@ -1578,8 +1617,10 @@ unsigned long memctlr_shrink_mapped_memo
   }
  }
 out:
+ current->reclaim_state = NULL;
  return ret;
 }
+
 #endif

 /* It's optimal to keep kswapds on the same CPUs as their memory, but

--
```

Subject: Re: [PATCH 3/3][RFC] Containers: Pagecache controller reclaim
Posted by Vaidyanathan Srinivas on Tue, 27 Mar 2007 07:17:10 GMT
View Forum Message <> Reply to Message

```
Aubrey Li wrote:
> On 3/6/07, Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com> wrote:
>> The reclaim code is similar to RSS memory controller.  Scan control is
>> slightly different since we are targeting different type of pages.
>>
>> Additionally no mapped pages are touched when scanning for pagecache pages.
>>
>> RSS memory controller and pagecache controller share common code in reclaim
>> and hence pagecache controller patches are dependent on RSS memory controller
>> patch even though the features are independently configurable at compile time.
>>
>> --- linux-2.6.20.orig/mm/vmscan.c
>> +++ linux-2.6.20/mm/vmscan.c
>> @@ -43,6 +43,7 @@
>>
>>  #include <linux/swapops.h>
>>  #include <linux/memcontrol.h>
>> +#include <linux/pagecache_acct.h>
>>
>>  #include "internal.h"
>>
>> @@ -70,6 +71,8 @@ struct scan_control {
>>
>>        struct container *container;    /* Used by containers for reclaiming */
>>                                        /* pages when the limit is exceeded  */
>> +	int reclaim_pagecache_only;	/* Set when called from
>> +					pagecache controller */
>>  };
>>
>>  /*
>> @@ -474,6 +477,15 @@ static unsigned long shrink_page_list(st
>> 			goto keep;
>>
>> 		VM_BUG_ON(PageActive(page));
>> +		/* Take it easy if we are doing only pagecache pages */
>> +		if (sc->reclaim_pagecache_only) {
>> +			/* Check if this is a pagecache page they are not mapped */
>> +			if (page_mapped(page))
>> +				goto keep_locked;
>> +			/* Check if this container has exceeded pagecache limit */
>> +			if (!pagecache_acct_page_overlimit(page))
>> +				goto keep_locked;
>> +		}
>>
>> 		sc->nr_scanned++;
>>
>> @@ -522,7 +534,8 @@ static unsigned long shrink_page_list(st
>> 		}
>>
>> 		if (PageDirty(page)) {
>> -			if (referenced)
>> +			/* Reclaim even referenced pagecache pages if over limit */
>> +			if (!pagecache_acct_page_overlimit(page) && referenced)
>> 				goto keep_locked;
>> 			if (!may_enter_fs)
>> 				goto keep_locked;
>> @@ -869,6 +882,13 @@ force_reclaim_mapped:
>> 		cond_resched();
>> 		page = lru_to_page(&l_hold);
>> 		list_del(&page->lru);
>> +		/* While reclaiming pagecache make it easy */
>> +		if (sc->reclaim_pagecache_only) {
>> +			if (page_mapped(page) || !pagecache_acct_page_overlimit(page)) {
>> +				list_add(&page->lru, &l_active);
>> +				continue;
>> +			}
>> +		}
>
> Please correct me if I'm wrong.
> Here, if page type is mapped or not overlimit, why add it back to active list?
> Did  shrink_page_list() is called by shrink_inactive_list()?
```

Correct, shrink_page_list() is called from shrink_inactive_list() but

the above code is patched in shrink_active_list(). The
'force_reclaim_mapped' label is from function shrink_active_list() and
not in shrink_page_list() as it may seem in the patch file.

While removing pages from active_list, we want to select only
pagecache pages and leave the remaining in the active_list.
page_mapped() pages are _not_ of interest to pagecache controller
(they will be taken care by rss controller) and hence we put it back.
 Also if the pagecache controller is below limit, no need to reclaim
so we put back all pages and come out.

--Vaidy

Subject: Re: [PATCH 3/3][RFC] Containers: Pagecache controller reclaim
Posted by Aubrey Li on Tue, 27 Mar 2007 08:41:33 GMT
View Forum Message <> Reply to Message

On 3/27/07, Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com> wrote:
> Correct, shrink_page_list() is called from shrink_inactive_list() but
> the above code is patched in shrink_active_list(). The
> 'force_reclaim_mapped' label is from function shrink_active_list() and
> not in shrink_page_list() as it may seem in the patch file.
>
> While removing pages from active_list, we want to select only
> pagecache pages and leave the remaining in the active_list.
> page_mapped() pages are _not_ of interest to pagecache controller
> (they will be taken care by rss controller) and hence we put it back.
> Also if the pagecache controller is below limit, no need to reclaim
> so we put back all pages and come out.

Oh, I just read the patch, not apply it to my local tree, I'm working
on 2.6.19 now.
So the question is, when vfs pagecache limit is hit, the current
implementation just reclaim few pages, so it's quite possible the
limit is hit again, and hence the reclaim code will be called again
and again, that will impact application performance.

-Aubrey

Subject: Re: [PATCH 3/3][RFC] Containers: Pagecache controller reclaim
Posted by Vaidyanathan Srinivas on Tue, 27 Mar 2007 09:44:57 GMT
View Forum Message <> Reply to Message

Aubrey Li wrote:
> On 3/27/07, Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com> wrote:

>> Correct, shrink_page_list() is called from shrink_inactive_list() but
>> the above code is patched in shrink_active_list(). The
>> 'force_reclaim_mapped' label is from function shrink_active_list() and
>> not in shrink_page_list() as it may seem in the patch file.
>>
>> While removing pages from active_list, we want to select only
>> pagecache pages and leave the remaining in the active_list.
>> page_mapped() pages are _not_ of interest to pagecache controller
>> (they will be taken care by rss controller) and hence we put it back.
>> Also if the pagecache controller is below limit, no need to reclaim
>> so we put back all pages and come out.
>
> Oh, I just read the patch, not apply it to my local tree, I'm working
> on 2.6.19 now.
> So the question is, when vfs pagecache limit is hit, the current
> implementation just reclaim few pages, so it's quite possible the
> limit is hit again, and hence the reclaim code will be called again
> and again, that will impact application performance.

Yes, you are correct. So if we start reclaiming one page at a time,
then the cost of reclaim is very high and we would be calling the
reclaim code too often. Hence we have a 'buffer zone' or 'reclaim
threshold' or 'push back' around the limit. In the patch we have a 64
page (256KB) NR_PAGES_RECLAIM_THRESHOLD:

```
 int pagecache_acct_shrink_used(unsigned long nr_pages)
 {
  unsigned long ret = 0;
  atomic_inc(&reclaim_count);
+
+ /* Don't call reclaim for each page above limit */
+ if (nr_pages > NR_PAGES_RECLAIM_THRESHOLD) {
+  ret += shrink_container_memory(
+    RECLAIM_PAGECACHE_MEMORY, nr_pages, NULL);
+ }
+
  return 0;
 }
```

Hence we do not call the reclaimer if the threshold is exceeded by
just 1 page... we wait for 64 pages or 256KB of pagecache memory to go
 overlimit and then call the reclaimer which will reclaim all 64 pages
in one shot.

This prevents the reclaim code from being called too often and it also
keeps the cost of reclaim low.

In future patches we are planing to have a percentage based reclaim threshold so that it would scale well with the container size.

--Vaidy

Subject: Re: [PATCH 3/3][RFC] Containers: Pagecache controller reclaim
Posted by Aubrey Li on Tue, 27 Mar 2007 10:53:39 GMT
View Forum Message <> Reply to Message

```
On 3/27/07, Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com> wrote:
>
>
> Aubrey Li wrote:
> > On 3/27/07, Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com> wrote:
> >> Correct, shrink_page_list() is called from shrink_inactive_list() but
> >> the above code is patched in shrink_active_list().  The
> >> 'force_reclaim_mapped' label is from function shrink_active_list() and
> >> not in shrink_page_list() as it may seem in the patch file.
> >>
> >> While removing pages from active_list, we want to select only
> >> pagecache pages and leave the remaining in the active_list.
> >> page_mapped() pages are _not_ of interest to pagecache controller
> >> (they will be taken care by rss controller) and hence we put it back.
> >>  Also if the pagecache controller is below limit, no need to reclaim
> >> so we put back all pages and come out.
> >
> > Oh, I just read the patch, not apply it to my local tree, I'm working
> > on 2.6.19 now.
> > So the question is, when vfs pagecache limit is hit, the current
> > implementation just reclaim few pages, so it's quite possible the
> > limit is hit again, and hence the reclaim code will be called again
> > and again, that will impact application performance.
>
> Yes, you are correct.  So if we start reclaiming one page at a time,
> then the cost of reclaim is very high and we would be calling the
> reclaim code too often.  Hence we have a 'buffer zone' or 'reclaim
> threshold' or 'push back' around the limit.  In the patch we have a 64
> page (256KB) NR_PAGES_RECLAIM_THRESHOLD:
>
>  int pagecache_acct_shrink_used(unsigned long nr_pages)
>  {
>       unsigned long ret = 0;
>       atomic_inc(&reclaim_count);
> +
> +      /* Don't call reclaim for each page above limit */
> +      if (nr_pages > NR_PAGES_RECLAIM_THRESHOLD) {
> +              ret += shrink_container_memory(
> +                              RECLAIM_PAGECACHE_MEMORY, nr_pages, NULL);
```

> +   }
> +
>   return 0;
> }
>
> Hence we do not call the reclaimer if the threshold is exceeded by
> just 1 page... we wait for 64 pages or 256KB of pagecache memory to go
> overlimit and then call the reclaimer which will reclaim all 64 pages
> in one shot.
>
> This prevents the reclaim code from being called too often and it also
> keeps the cost of reclaim low.
>
> In future patches we are planing to have a percentage based reclaim
> threshold so that it would scale well with the container size.
>
Actually it's not a good idea IMHO. No matter how big the threshold
is, it's not suitable. If it's too small, application performance will
be impacted seriously after pagecache limit is hit. If it's too large,
Limiting pagecache is useless.

Why not reclaim pages as much as possible when the pagecache limit is hit?

-Aubrey

Subject: Re: [PATCH 3/3][RFC] Containers: Pagecache controller reclaim
Posted by Vaidyanathan Srinivas on Tue, 27 Mar 2007 12:25:06 GMT
View Forum Message <> Reply to Message

Aubrey Li wrote:
> On 3/27/07, Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com> wrote:
>>
>> Aubrey Li wrote:
>>> On 3/27/07, Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com> wrote:
>>>> Correct, shrink_page_list() is called from shrink_inactive_list() but
>>>> the above code is patched in shrink_active_list(). The
>>>> 'force_reclaim_mapped' label is from function shrink_active_list() and
>>>> not in shrink_page_list() as it may seem in the patch file.
>>>>
>>>> While removing pages from active_list, we want to select only
>>>> pagecache pages and leave the remaining in the active_list.
>>>> page_mapped() pages are _not_ of interest to pagecache controller
>>>> (they will be taken care by rss controller) and hence we put it back.
>>>> Also if the pagecache controller is below limit, no need to reclaim
>>>> so we put back all pages and come out.
>>> Oh, I just read the patch, not apply it to my local tree, I'm working
>>> on 2.6.19 now.

>>> So the question is, when vfs pagecache limit is hit, the current
>>> implementation just reclaim few pages, so it's quite possible the
>>> limit is hit again, and hence the reclaim code will be called again
>>> and again, that will impact application performance.
>> Yes, you are correct. So if we start reclaiming one page at a time,
>> then the cost of reclaim is very high and we would be calling the
>> reclaim code too often. Hence we have a 'buffer zone' or 'reclaim
>> threshold' or 'push back' around the limit. In the patch we have a 64
>> page (256KB) NR_PAGES_RECLAIM_THRESHOLD:
>>
```
>>  int pagecache_acct_shrink_used(unsigned long nr_pages)
>>  {
>>         unsigned long ret = 0;
>>         atomic_inc(&reclaim_count);
>> +
>> +       /* Don't call reclaim for each page above limit */
>> +       if (nr_pages > NR_PAGES_RECLAIM_THRESHOLD) {
>> +               ret += shrink_container_memory(
>> +                               RECLAIM_PAGECACHE_MEMORY, nr_pages, NULL);
>> +       }
>> +
>>         return 0;
>>  }
```
>>
>> Hence we do not call the reclaimer if the threshold is exceeded by
>> just 1 page... we wait for 64 pages or 256KB of pagecache memory to go
>> overlimit and then call the reclaimer which will reclaim all 64 pages
>> in one shot.
>>
>> This prevents the reclaim code from being called too often and it also
>> keeps the cost of reclaim low.
>>
>> In future patches we are planing to have a percentage based reclaim
>> threshold so that it would scale well with the container size.
>>
> Actually it's not a good idea IMHO. No matter how big the threshold
> is, it's not suitable. If it's too small, application performance will
> be impacted seriously after pagecache limit is hit. If it's too large,
> Limiting pagecache is useless.
>
> Why not reclaim pages as much as possible when the pagecache limit is hit?
>

Well, that seems to be a good suggestion. We will try it out by
asking the reclaimer to do as much as possible in minimum time/effort.
 However we have to figure out how hard we want to push the reclaimer.
 In fact we can push the shrink_active_list() and
shrink_inactive_list() routines to reclaim the _all_ container pages.

We do have reclaim priority to play with. Let see if we can comeup
with some automatic method to reclaim 'good' number of pages each time.

--Vaidy