Resizing Memory With Balloons and Hotplug

Joel H. Schopp
IBM
jschopp@austin.ibm.com

Keir Fraser
University of Cambridge Computer Labratory
Keir.Fraser@cl.cam.ac.uk

Martine J. Silbermann
HP
martine.silbermann@hp.com

Abstract

In a virtualized environment it is often necessary to resize the amount of memory allocated to a particular copy of Linux. There are currently two viable approaches to adding and removing memory: memory hotplug and a balloon driver. We will compare and contrast how these resizing technologies work independently, weighing the benefits and drawbacks of each one. We also will show how these two resizing technologies could be used together to provide the best of both worlds.

1 Introduction

It is often the case that Linux is not running on real hardware. Instead Linux is running in a virtualized enviornment such as Xen[2], VirtualPC, pSeries, zVM, or VirtualIron[1]. In these environments a premium is put on efficient utilization of resources by balancing the use of these resources during run time. Some of these systems may partition resources, others may fully virtualize them and assign real resources dynamically. However, Linux still behaves as if it were running on real, non-changing, hardware by itself. This real hardware mentality means that users of Linux have to reboot in order to change while other operating systems in the same environments do not miss a beat. In many environments where reboot is impractical Linux will have bad performance. It will have bad performance because it will not be able to utilize extra memory made available after boot, and bad performance because it cannot decrease its view of memory to closely match decreases in the underlying memory it actually has.

2 Motivation

The question for Linux is not if it will change to accomodate new virtualized environments, but how and when Linux will change. If the Linux community wants to have Linux grow in the enterprise market we need to address some of our missing features, an important one of which is resizing memory.

3 Memory Hotplug Add

3.1 How Hotplug Add Works

Memory hotplug add works as if a physical DIMM were added. The firmware, ACPI or pHYP for example, tells the OS a new address range of memory is available. The size of this new range of memory must be a multiple of the `SECTION` size in `CONFIG_SPARSEMEM`. On powerpc the section size is 16MB, so 32MB or 64MB could be added, but 40MB could not be. After Linux finds the new memory it sets up a new `mem_map[]` and other structures. Finally the kernel then adds the new memory into the allocator, making it available for use.

3.2 Current Status

Memory hotplug add is in mainline as of 2.6.14. There is a dependency on `CONFIG_SPARSEMEM`[3], which is also in mainline. Hotplug add also required changes to the buddy allocator[3]. Memory hot add works well in NUMA and non-NUMA systems. In NUMA systems new memory is added as if it were in existing nodes. Code to online new nodes was submitted on March 17th, 2005 by Yasunori Goto. Current Distros SLES10 and RHEL5 have hotplug add enabled in powerpc kernels.

3.3 Hotplug Add Advantages & Disadvantages

The biggest advantage hotplug add has is that it is in the mainline tree. Because it is in mainline the code remains up to date, and any kernel based on mainline needs only be configured on at compile time in order to use hotplug add. Most major architectures are supported, and much of the code is common to the architectures making it easy to add new architectures. Memory hotplug add solves the problem of adding memory that wasn't present at boot to scale Linux up in response to changing resources.

Hoptlug add has a dependency on `CONFIG_SPARSEMEM`, which is still relatively new and has not completely replaced `CONFIG_DISCONTIGMEM`. Over time `CONFIG_SPARSEMEM` is expected to fully replace `CONFIG_DISCONTIGMEM` on i386, x86-64, and ia-64 as it works well in mainline on all supported architectures. However, as of this writing Novell and Redhat have only enabled `CONFIG_SPARSEMEM` in their powerpc kernels. Another major disadvantage of memory hotplug add is its compile time fixed granularity which takes away some flexibility from smaller environments. This is because the section size is set at compile time and must be acceptable on both large servers and small memory systems, but is primarily targeted at larger memory systems.

4 Memory Hotplug Remove

4.1 How Hotplug Remove Works

In memory hotplug remove all the pages in some memory section must be freed in a timely fashion. Simultaneously, processes are running as usual, and need to continue running with good performance. It is thus necessary to move the memory in the targeted section to a new location safely and quickly. This is known as memory migration. By removing unallocated pages from the allocator and migrating all allocated pages with memory migration it is thus possible to completely empty all data from an entire section. Once a memory section is empty

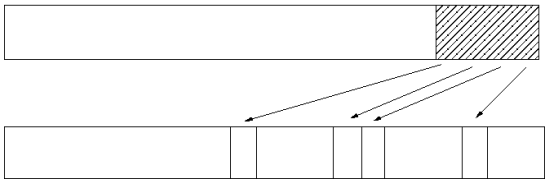

Figure 1: Memory remove uses migration to remove contiguous sections

all references to it can be removed. After it is no longer referenced by the kernel memory manager it can be safely removed as if it were not there to begin with.

Not all memory is directly migrateable. The Linux kernel has a constant offset between virtual addresses and physical addresses. It also caches some of the physical addresses. If any of this non-migrateable memory is located in the section targeted for removal, then the whole section cannot be removed. Some of this memory is reclaimed by running `shrink_list()`. Other memory associated with filesystems has filesystem callbacks. But unless the Linux community decides to have a remappable kernel there will always be some amount of memory that cannot be migrated.

4.2 Current Status

Memory Hotplug remove is not in mainline. Patches exist, released under the GPL, but are only occasionally rebased. To be worthwhile the existing patches would need either a remappable kernel, which remains highly doubtful, or a fragmentation avoidance strategy to keep migrateable and non-migrateable pages clumped together nicely. Two patch sets exist to do fragmentation avoidance, the details of which are in a paper by Mel Gorman and Andy Whitcroft[4].

4.3 Hotplug Remove Advantages & Disadvantages

In hotplug remove the sections being removed are large and contiguous so they don't cause any external fragmentation. The kernel also has an accurate view of how much memory it really has, leaving kernel developers the ability to be smart instead of lucky about managing memory efficiently. Hoptlug remove is in many ways an ideal solution.

However, hotplug remove is not without disadvantages. It faces much opposition to being integrated in mainline kernels because of its dependency on the fragmentation avoidance. This opposition comes from the fact that fragmentation avoidance modifies key kernel components, making them more complex. Furthermore, memory hotplug as currently designed has limitations on not being able to remove memory containing certain kinds of allocations. This limitation makes it less useful for physical removal of DIMMs or of predictive disabling of failing memory because of the likelyhood of that memory containing non-removable allocations. For these and other reasons too lengthy to present, hotplug remove's major disadvantage is that it will be a long process of community feedback and development before it becomes available in mainline kernels or from distributors.

5 Balloon Drivers

5.1 How Balloon Drivers Work

Balloon drivers have been used by virtualization as a means to manage memory in a multiple virtual machines environment. The idea

is very simple and offers the advantage of being minimally invasive to the guest OS. However, it does require collaboration from this guest in order for it to be effective. In 2002 Waldspurger[5] introduced the concept of ballooning as it was implemented in the VMware ESX Server. In 2003 this concept was adopted by the Xen team[6] to support their memory management needs. The concept used by the two virtualization approaches is the same, but the terminology used in the implementation descriptions vary. Therefore, we will use neutral terminology in our description.

The basic function of the balloon driver is to pass memory pages back and forth between the hypervisor and the virtual machine page allocator. This provides a solution for load-balancing and also addresses the issue of memory oversubscription. When a virtual machine is created a range of permissible amounts of memory allocation is specified. The lower bound of that range corresponds to the minimal amount of memory under which the virtual machine can reasonably operate; this is loosely defined as being able to operate without excessive swapping. The upper bound of that range corresponds to the maximum amount of memory that will ever be allocated to this VM no matter how much memory the hypervisor has available.

The balloon driver resides in the VM but is controlled by the hypervisor. When the balloon inflates creating memory pressure in the VM the memory management routines of the guest OS must reclaim space to satisfy the driver allocation request. When memory is tight that might necessitate that the guest OS decides which pages to reclaim, possibly swapping those to its own virtual disk. The reclaimed pages are passed down to the hypervisor which in turn makes the physical memory available to other VMs. In order to guarantee separation between VMs the page are zeroed out before being made available to other VMs. When the balloon deflates the memory is made available again for general use by this guest OS. Since the balloon driver inflates by allocating memory in the VM it is obvious that without the collaboration of the guest OS this technique is not very successful in releasing memory to the hypervisor. However, when the collaboration works then ballooning can be a very effective and predictable way to positively affect performance in an environment where workloads benefit from additional memory[5].

Different mechanisms for requesting changes in the size of the balloon are used in the two implementations: in VMware the balloon driver polls the server once per second for changes in the balloon size, while Xen uses a mechanism relying on the XenStore/XenBus functionality. XenBus provides a bus abstraction for paravirtualized drivers to communicate between domains and XenStore is a filesystem-like database that is accessible by all domains. Most commonly, management tools configure and control virtual devices by writing values into keys in the database that trigger events in drivers. In the case of the balloon driver the target size of the balloon is stored in a key and the balloon driver sets a watch on it. When the value of the key changes the driver immediately responds by trying to accomodate to the requested size. Neither one of those mechanisms has a significant impact in terms of performance to the VM.

5.2 Balloon Drivers Advantages & Disadvantages

It is sometimes difficult to separate the advantages and disadvantages of this technique, since what could be viewed as an advantage could also be viewd as a limitation. For example, it is a definite advantage to be able to directly use the native memory management routines of

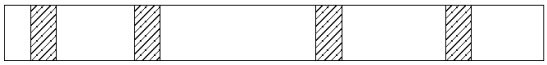

Figure 2: Balloon Driver removes memory that is already free, usually causing fragmentation

the guest OS without any changes. However, the use of existing mememory managment routines becomes a real limitation when you are trying to maintain low external memory fragmentation.

Also, the balloon driver is a very independent entity in the VM whose size is only known by itself and the hypervisor; there is currently no notification to the applications that the balloon driver is squeezing memory. Moreover memory statistics reporting tools such as `top` or `free` will see no changes in the memory usage since from the kernel's perspective the memory is used by the balloon driver.

Furthermore, even if the guest VM is trying to collaborate the balloon driver might not be able to reclaim the memory requested by the hypervisor fast enough to satisfy the system's needs.

The most obvious disadvantages of ballooning is that the balloon driver fragments the pseudophysical memory map of the guest VM. Although hotplug memory is more invasive, it is able to alloc/dealloc contiguous regions of memory. Thus it avoids fragmentation of the memory map. Instead, the memory map can be split up into sections using `SPARSEMEM` and hoptlug can alloc/free whole sections at a time, freeing memory metadata at the same time you free the memory itself. It also potentially makes it easier to implement compacting of real physical memory, so that Xen itself does not end up with excessive fragmented memory.

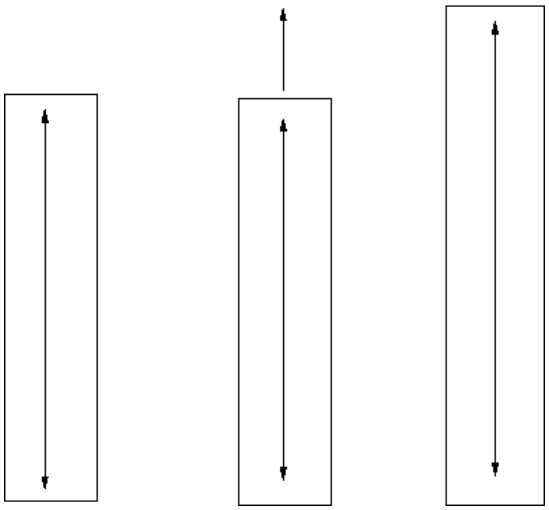

Figure 3: From left to right: System with balloon driver, hotplug add operation, The same system after hotplug add operation

6 Steps Towards Unification

Memory hotplug and balloon drivers serve similar goals in different ways. They are like peanut butter and jelly. Either one can be used on its own, but if they are used together in proper balance the end result can be better than either alone.

6.1 Balloon Driver Plus Add

Balloon drivers are already in use in virtualized environments, but are unable to use more memory than the Linux kernel was booted with. At the same time memory hotplug add is already in mainline kernels and allows Linux to add memory that it wasn't booted with. Without too much code the existing balloon driver could be modified to grow memory past normal balloon limits using the hotplug add mechanism. The small amount of new code would be contained in the balloon driver, making it easy to merge into mainline.

6.2 Balloon Driver Plus Remove

The reason hotplug remove is not merged into mainline yet is because without fragmentation avoidance it would not have a high rate of success removing whole sections, making it not reliable enough to use on its own. However, using existing memory migration and just enough code to enable removing sections does enable the removing of sections in good conditions. Again, using a balloon driver as a base it is possible to add hotplug remove into the mix. The balloon driver could, with a minimal amount of extra code, use hotplug remove as a primary vehicle and when hotplug remove is unable to free a whole section it could fall back to its normal methodology.

This combination would improve on the fragmentation of the normal balloon allocator whenever hotplug remove was used. It also alleviates the dependence for fragmentation avoidance, but would still benefit from any fragmentation avoidance that was eventually merged.

6.3 Make Balloon Memory Migrateable

One of the obstacles to using memory remove in combination with a balloon driver is that memory already taken away by the balloon driver is unable to be removed a second time by hotplug. This is easily alleviated by two simple steps. The first step is to mark the pages already taken by the balloon driver as migrateable. The second step is to add event notifiers in the balloon driver that would release its claim on any memory targeted for removal.

By doing these two steps the hotplug remove mechanism could remove sections that had been fragmented by the balloon driver. This would also clean up data structures associated with that memory.

7 Conclusion

In an ideal world memory hotplug remove would be the primary memory management interface and would work on all memory. But even without it there is a lot that can be done with existing and easily merged technology to help make Linux a more flexible OS in a virtualized environment.

8 Acknowledgments

Many thanks to Yasunori Goto, Dave Hansen, Mike Kravetz, Hirokazu Takahashi, IWAMOTO Toshihiro, KAMEZAWA Hiroyuki, Matt Tolentino, Bob Picco, Andy Whitcroft, and all the hotplug developers.

9 Legal Statement

References

[1] `http://www.virtualiron.com`

[2] `http://www.xensource.com`

[3] J. Schopp et al. Memory Hotplug Redux. In *Proceedings of the Ottawa Linux Symposium,*, Ottawa, Ontario, Canada, July 2005.

[4] M. Gorman. The What, They Why and the Where To of Anti Fragmentation. In *Proceeding sof the Ottawa Linux Symposium,*, Ottawa, Ontario, Canada, July 2006.

[5] C. A. Waldspurger, Memory resource management in VMware ESX server, Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002), ACM Operating Systems Review, Winter 2002 Special Issue, pages 181-194, Boston, MA, USA, Dec. 2002

[6] Paul Barham, Boris Dragovic, Keir Fraser, Steven Han, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield, Xen and the Art of Virtualization, Proceedings of the 19th ACM Symposium on Operating Systems Principles, October 19-22, 2003

Proceedings of the Linux Symposium

Volume Two

July 19th–22nd, 2006
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*

Review Committee

Jeff Garzik, *Red Hat Software*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat Software*
Ben LaHaise, *Intel Corporation*
Matt Mackall, *Selenic Consulting*
Patrick Mochel, *Intel Corporation*
C. Craig Ross, *Linux Symposium*
Andrew Hutton, *Steamballoon, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
David M. Fellows, *Fellows and Carr, Inc.*
Kyle McMartin