HugeTLB Improvement to Reduce Post-Boot Allocation Failures

Joshua Nicolas Verburg-Sachs

2016-02

Contents

1 Preamble

This research was inspired by many frustrated attempts to utilize the excellent huge page facilities of the Linux kernel during runtime. Huge pages are a mechanism whereby the Linux kernel can allocate memory not in 4kb pages (or whatever your distribution uses) but page sizes ranging from many megabytes to a gigabyte or more (though we touch only on the megabyte sized pages in this paper). This gives a variety of advantages, one of the most important being a reduction in the number of page entries that the kernel must manage for your executable. This can, for example, significantly increase the speed of large mmaps and munmaps.

Most of the current use cases and documentation stipulate that you should only attempt to allocate huge pages immediately after boot, in order to avoid (very common) failures due to memory fragmentation and memory use by other executables. However this is not always a convenient option and in some situations (such as real time systems with uptime requirements) it is basically untenable.

Therefore, this paper will describe a method whereby the Linux kernel can be improved to more robustly allocate huge pages during runtime, even when other applications have consumed a significant amount of available memory.

2 Amble

There are two primary causes to hugepage allocation failure. In order to illuminate them and discuss the solution we will describe the original algorithm here: **??**

3 Mosey

Below are two figures which describe 6 different scenarios for the default and new allocation method (hereafter called aggressive). The scenarios are simply a set of configuration options for the test run in question. These options are as follows:

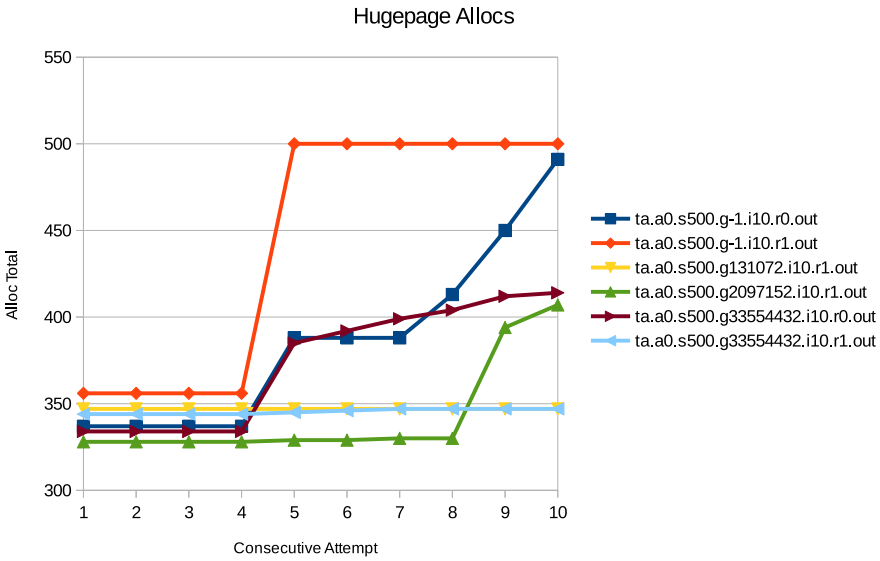
- Aggressive: Whether to use aggressive alloc.
- Set: The number of pages to allocate.
- Grab: Whether to cause the test application to allocate memory (to disrupt hugepage allocation), and how much if so.
- Iterations: How many times the test application should attempt to set the number of huge pages. In all test runs this was 10, as it proved sufficient for demonstrating performance.
- Reset: Whether to re-set the number of hugepages to 0 after each iteration.

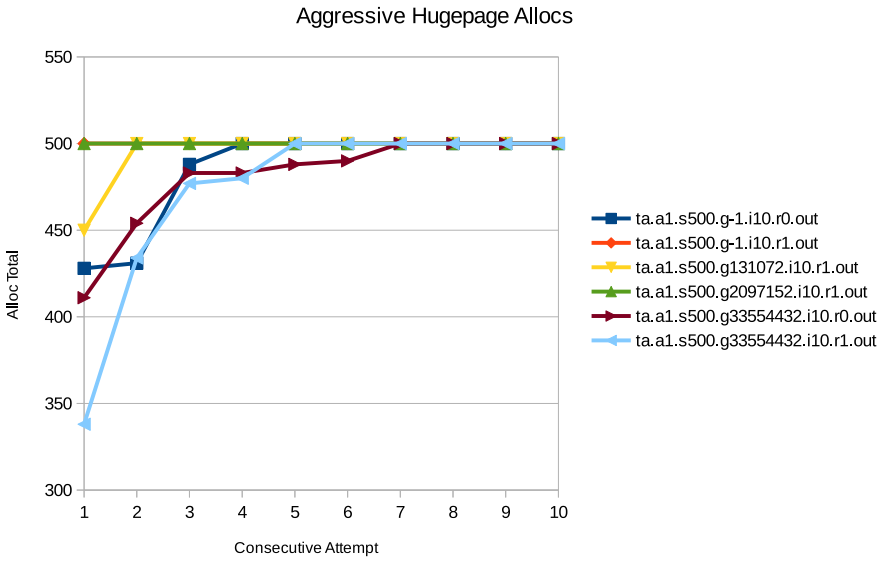
```
static int alloc_fresh_huge_page(struct hstate *h, nodemask_t *nodes_allowed)
{
        struct page *page;
        int nr_nodes, node;
        int ret = 0;

        for_each_node_mask_to_alloc(h, nr_nodes, node, nodes_allowed) {
                page = alloc_fresh_huge_page_node(h, node);
                if (page) {
                        ret = 1;
                        break;
                }
        }

        if (ret)
                count_vm_event(HTLB_BUDDY_PGALLOC);
        else
                count_vm_event(HTLB_BUDDY_PGALLOC_FAIL);

        return ret;
}
```





3

3

4 Postamble

5 Future Works

Appendix 1: Table

A Data Table

Table 1: Hugepage Allocation Data Points

Appendix 2: List of Figures

List of Figures