Kernel Optimization: Modifying Multiple Tasks Related Variables

Authors: Lázaro Cárdenas Guerrero & Alfonso Ruiz Velasco Ramírez

October 16 2016

1 Abstract

This document describes some kernel and memory variables, the fact that they affect how the system performs as a whole, and shows the results of some testing while modifying the variables described.

For an easier visualization we made some tables and graphs with the results and because of that our conclusions can be explained easier and clearer. We ended up thinking that testing is harder than it seems, and it takes time, patience and dedication to actually discover an improvement on the system.

2 Introduction

The operating system is the core of a computer. It manages its resources, and serves as an interface between the users and the computer. However, it is also tasked with managing the processes and deciding which one shall run next. This is made possible by the scheduler, a program in charge of loading processes into main memory and the CPU.

The scheduler is a complex program full of different variables that affect the computer's performance, such as latency, minimum granularity, migration cost, among others. It is important to understand how the scheduler behaves based on the value of these variables.

With this knowledge we could find ways to improve the scheduler of the OS based only on the values of each variable. An optimization on performance of the already existing scheduler could be found if the correct values are modified.

In order to do so, a stress test is to be applied several times. This with the objective of observing the performance of the scheduler when it is overloaded with tasks. The methodology to do this is explained on the development section.

3 Theoretical Framework

The kernel is composed about many things, so many that maybe changing one single variable won't change the end result that much, but, every little bit of information forcefully has to make an impact on the final product, no matter if it's a tiny change, it makes it different from how it would be without it.

In this section you'll get to know the variables that were modified during the experiments. Most of them are kernel variables:

- kernel.sched_nr_migrate: This variable is used to balance the ammount of tasks that will be sent to other CPUs to balance the workload between cores. This variable specifically states the number of tasks that will be moving at a time. Increasing the value of this variable will lead to an improvement in performance for threads that spawn a lot of tasks, but at the expense of realtime latencies.[1]

- kernel.sched_min_granularity_ns: This variable refers to the minimum amount a time a process needs to run before it is preempted.[2]

- kernel.sched_migration_cost_ns: This variable determines how long a migrated process has to be running before the kernel will consider migrating it again to another core. Setting this variable to the right value can lead to an improvement when handling many connections in a server.[3]

- kernel.sched_latency_ns: It is the time that happens between the moment when the kernel gets a signal about an interrupt or event happening, and the moment when the scheduler has the opportunity to schedule the waiting thread or process related to the event.[4]

But there is also a memory variable that was tested:

- vm.swappiness: This is a variable with a ranged value from 0 to 100 that controls the degree to which the system favors anonymous memory or the page cache. A high value improves file-system performance, while aggressively swapping less active processes out of physical memory. A low value tries not to swap processes out of memory, which usually decreases latency, at the cost of I/O performance.[1]

4 Objective

Our main goals while testing were to understand how the different variables in the kernel worked, and to find an improvement in the system as a whole, if this couldn't be achieved, we stated that our goal would be to justify the reasons that made us jump to that conclusion.

5 Justification

We started doing this benchmark because we wanted to know how hard could it be to find an improvement in performance, the variables we chose weren't chosen at random, those variables include the number of tasks that can be moved at a time, how much should we swap , how much should we wait to move a task from one core to another, or to preempt it, and how does that affects latency as a whole. The stress test we applied works perfectly fine with this variables, as it puts a big workload onto the processor, and makes the kernel scheduler work really hard to keep up with the big amount of requests, with an almost full workload we thought that the kernel scheduler would split tasks

among the cores, and by that, it would show different results depending on the specifications we gave it about dividing the tasks.

If we had succeeded, we would have discovered a way to make a computer work faster, and right about now, the people want faster computers, but also they don't want them to be more expensive, or bigger, working on a computer by improving it's software makes everyone save money on the hardware because of an optimization in the logic behind everything, doing this also leads to an evolution of the software, of how problems are handled and solved, with better techniques, computers will be able to solve more complex problems with the same resources and in less time.

1. **This is how we use bold letters**:
2. **2nd item in the list**:

6 Development

We selected and changed the variables mentioned before and we tested them with stress.io from the software phoronix. Each variable was tested 4 times with a modified value and 1 time with the default one, we did that to see (insert something here).

Each test was documented and the results table can be found in the Results section. From the table, a graph was made for each variable, comparing its value against (insert what we are comparing).

It is important to note that each test took between 7 and 11 minutes, so the total amount of time taken to capture information was about 3:30 hours. Look forward on the next section for the obtained results.

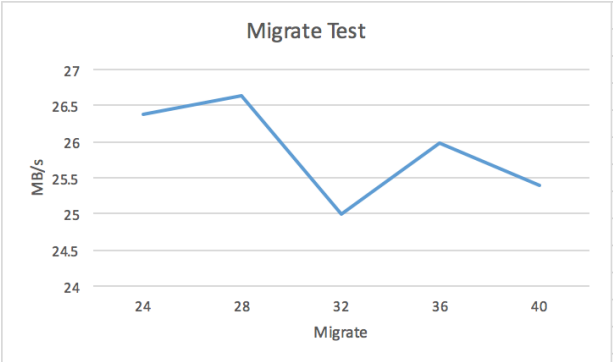
41 lines 9 paragraphs.

7 Results

After doing the stress.io test, we got the results that we show in the tables below.

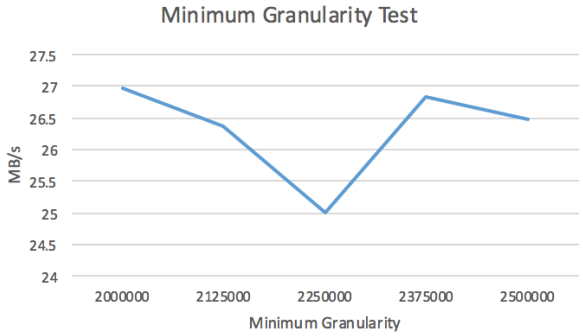
Test	kernel.sched_nr_migrate	MB/s
#1	24	26.38
#2	28	26.65
#3	32 (default)	25.01
#4	36	25.98
#5	40	25.39

Table 1: kernel.sched_nr_migrate test



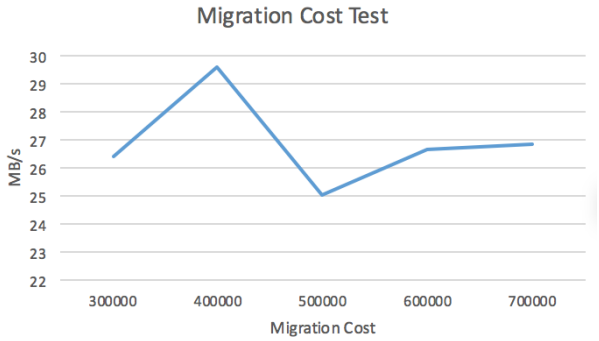
Test	kernel.sched_min_granularity_ns	MB/s
#1	2000000	26.98
#2	2125000	26.38
#3	2250000 (default)	25.01
#4	2375000	26.82
#5	2500000	26.48

Table 2: kernel.sched_min_granularity_ns test



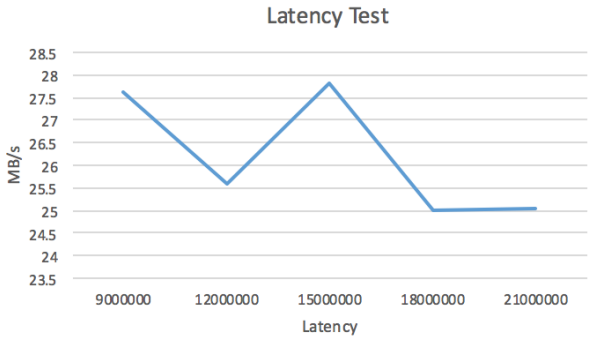
Test	kernel.sched_migration_cost_ns	MB/s
#1	300000	26.40
#2	400000	29.62
#3	500000 (default)	25.01
#4	600000	26.69
#5	700000	26.83

Table 3: kernel.sched_migration_cost_ns test



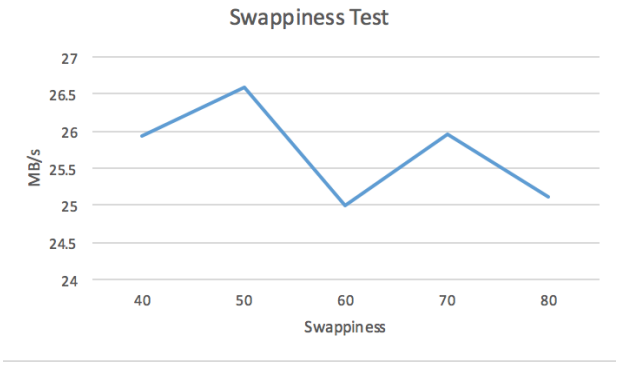
Test	kernel.sched_latency_ns	MB/s
#1	9000000	27.61
#2	12000000	25.58
#3	15000000	27.82
#4	18000000 (default)	25.01
#5	21000000	25.04

Table 4: kernel.sched_latency_ns test



Test	vm.swappiness	MB/s
#1	40	25.93
#2	50	26.59
#3	60 (default)	25.01
#4	70	25.95
#5	80	25.12

Table 5: vm.swappiness test



8 Conclusion

With the results it can be concluded that even though the variables were modified considerably, the results didn't show any important changes on performance. Therefore, it can be assumed that finding modifications that result on improvements on the scheduler can be a very difficult task.

For this reason a lot of testing is required to improve an operating system's performance based on modification of the scheduler and memory variables. Automatized tests would be more efficient than human managed tests, because a lot of time is needed to experiment with the biggest spectrum of values possible. But there should always be someone who'll check the end results of the tests, and the upsides/downsides of the changes needed to achieve those results (extra memory usage, latency, etc).

9 References

References

[1] Red Hat.(2016).Red Hat Enterprise MRG 1.2 Realtime Tuning Guide. Advanced tuning procedures for the Realtime component of Red Hat Enterprise MRG. Retrieved from: `https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_MRG/1.2/html/Realtime_Tuning_Guide/sect-Realtime_Tuning_Guide-Realtime_Specific_Tuning-Using_sched_nr_migrate_to_limit_SCHED_OTHER_processes..html`

[2] openSUSE.(2011).System Analysis and Tuning Guide. Retrieved from: `https://doc.opensuse.org/documentation/html/openSUSE_121/opensuse-tuning/cha.tuning.taskscheduler.html`

[3] Tweaked.(2012).The GNU/Linux Kernel. Retrieved from: `https://tweaked.io/guide/kernel/`

[4] EE Times.(2002). Linux scheduler latency. Retrieved from: `http://www.eetimes.com/document.asp?doc_id=1200916`