Solving N-Queens

Rodolfo Lepe & Gerardo Velasco

1630048 & 1225061

Abstract

Through the modification of the kernel on a Ubuntu system, we managed to solve the n-Queens problem after changing the default time-slice, swappiness, latency and wakeup-granularity to different values and testing the problem.

1 Introduction

Linux utilizes the Completely Fair Scheduling or CFS algorithm, which is an implementation of the Weighted fair queueing or WFQ. It consists of the CPU system starting with CFS time-slices the total CPU in runnable threads. The CFS looks to maximize the CPU utilization and the interactive performance.

The CFS scheduler calculates the timeslices for a task a moment before it is scheduled for its execution. Its lenght is variable and it totally depends on its priority and the load of the running queue.

The scheduler has the notion of every tasks virtually having the same runtime, but in reality the virtual runtime is a normalized value.

We will be running a process called n-queens, which is a benchmark that works with the famous problem of the 8 queens on a chess board.

2 Theoretical Framework

The computer we used was one with the following specs Memory 1.9 GB, Processor Intel Pentium (R) D CPU 2.80 GHz x 2, Graphics Gallium 0.4 on ATI RC410, OS Type 64-bit, Disk 490.1 GB. We used the Linux Ubuntu 16.04 operating system, it is a 64-bit OS. The Kernel is a 4.4.0-42-generic(x86_64), the desktop is the Unity 7.4.0, the display server is the X server 1.18.3. The displays driver radeon 7.7.0, an OpenGL:2.1 Mesa11-2-0 Gallium 0.4 Compiler: GCC 5.4.0 20160609, a File system: ext4, and it has a screen resolution of 1266x768. We tried to change some aspects of the Kernel so we could spot differences while running the benchmark called N-Queens.

The Swappiness in the Linux kernel is the responsible to control the relative weight for swapping out runtime memory. Its default value is 60, and setting it to a different value will affect other components such as latency. Latency is the interval between a stimulus and a response. Granularity depends on how many pieces can you distinguish from one another on a system. Timeslice is another name for preemtive multitasking.

We will be using an old computer, so if we break the Kernel we will not regret it as much. Our tests will be longer for the same reason, we will be using a CPU with Pentium, so patience is a key aspect.

3 Objective

To find a way by changing the values of several Kernel components to use a benchmark and generate a better overall process for it to be utilized. We will try to find a way to optimize Linux by making trials with the N-Queens benchmark.

4 Justification

The N-queens is one of the benchmarks that the code accepted, it is a simple program and we can try it several times with a low wait time. Every run takes approximately 5 minutes to load and it usually repeats it 3 times to get a better average time.

We know the basic concepts of how swappiness, latency, timeslice and granularity work, so we can change the values to expect different results.

N-queens is a simple algorithm that we already knew how it works, so we can have a deeper understanding of the overall situation.

5 Development

In order to test our theory, we used the platform Intel Pentium (R) D CPU 2.80 GHz x 2m which is kind of old, but it will do the trick just fine. The main characteristics of the computer are described in the Theoretical Framework.

The operating system that we used is the Ubuntu 16.04, the description of the system is listed in the Ubuntu project site home page (http://www.ubuntu..com).

We decided to use the benchmark called N-Queens. This benchmark solves the chess problem of putting N queens on an N x N board so that no queen can attack each other, this problem usually uses a recursive search for a placement of the queens that meets the correct conditions. The first reference of this problem is in a German magazine by Max Bezzel, this was on the year 1848, later it was studied by Gauss in the 1850, this after reading an article by Franz Nauck, who discovered all 92 solutions to the 8-Queen problem. Its most common use is for undergraduate students, because they need to understand and program a particular kind of algorithm.

The variables that we changed were the ones mentioned above, and we went all over the field with those variables. The table shows the initial values of those variables:

Variable	Initial value
vm.swappiness	40
kernel.sched_latency_ns	12000000
kernel.sched_wakeup_granularity_ns	2000000
kernel.sched_rr_timeslice_ns	25

6 Results

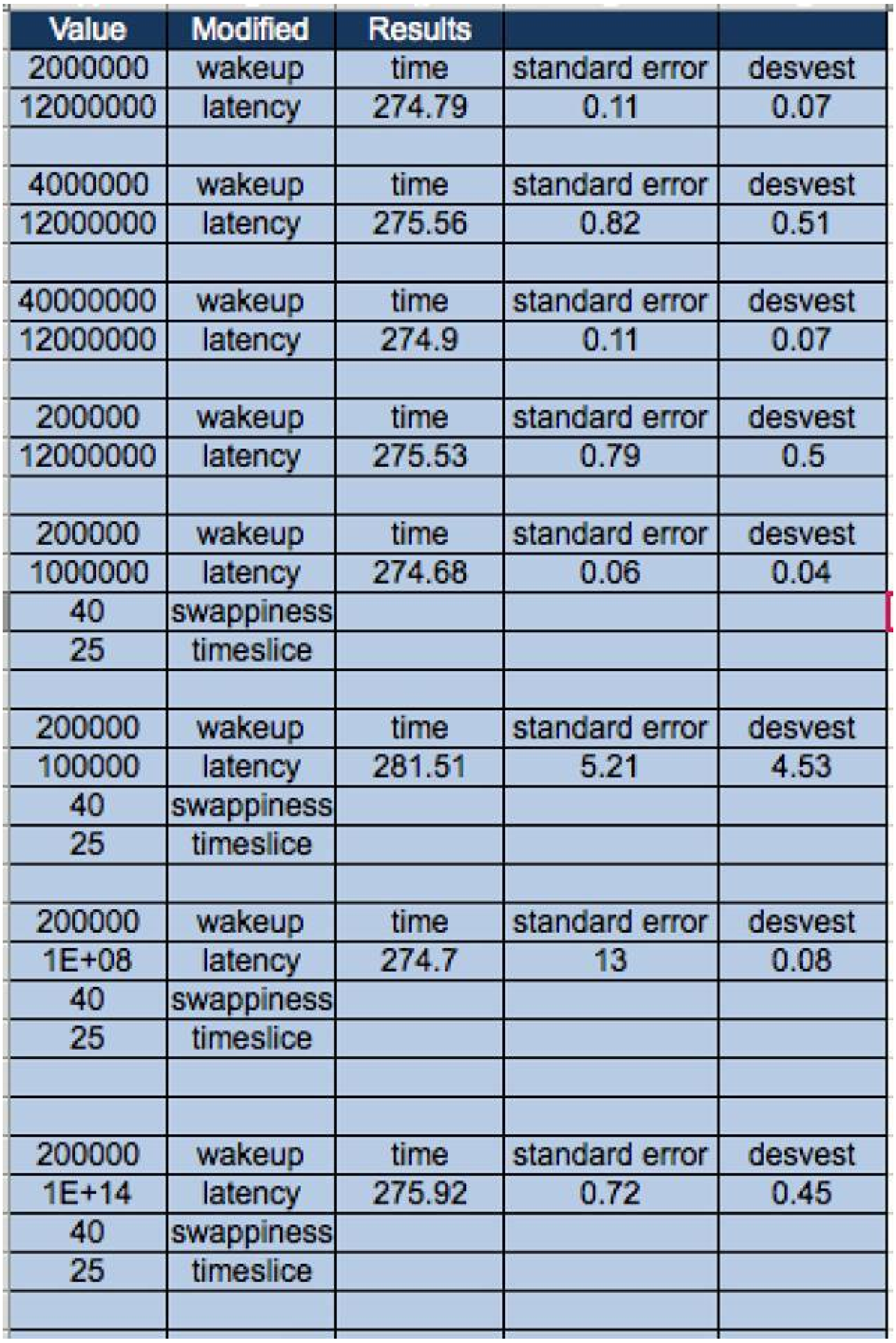
The results after modifying the different variables are shown below.

As seen in the tables and graph not much changed, you can almost say, that nothing really changed (except that one value that went of the charts, but we believe that was just a malfunction of the benchmark or the computer and not really a number that we care about) All the times are in seconds and de standard deviation is in percentages. If we take in consideration that result, we can see in the graphic that we have a function that looks pretty similar to a logarithmic function, but we cannot be sure because we don't have enough data. We went all over the board with the range of the variables, making all kind of different combinations, but it didn't matter how different they were, how much we tried, or how many times we did the same test, the time the process took to finish was almost the same, never varying for more than 1 second. We spent more than 6 hours testing and we could not find any result that varied a lot from the common results.

7 Conclusion

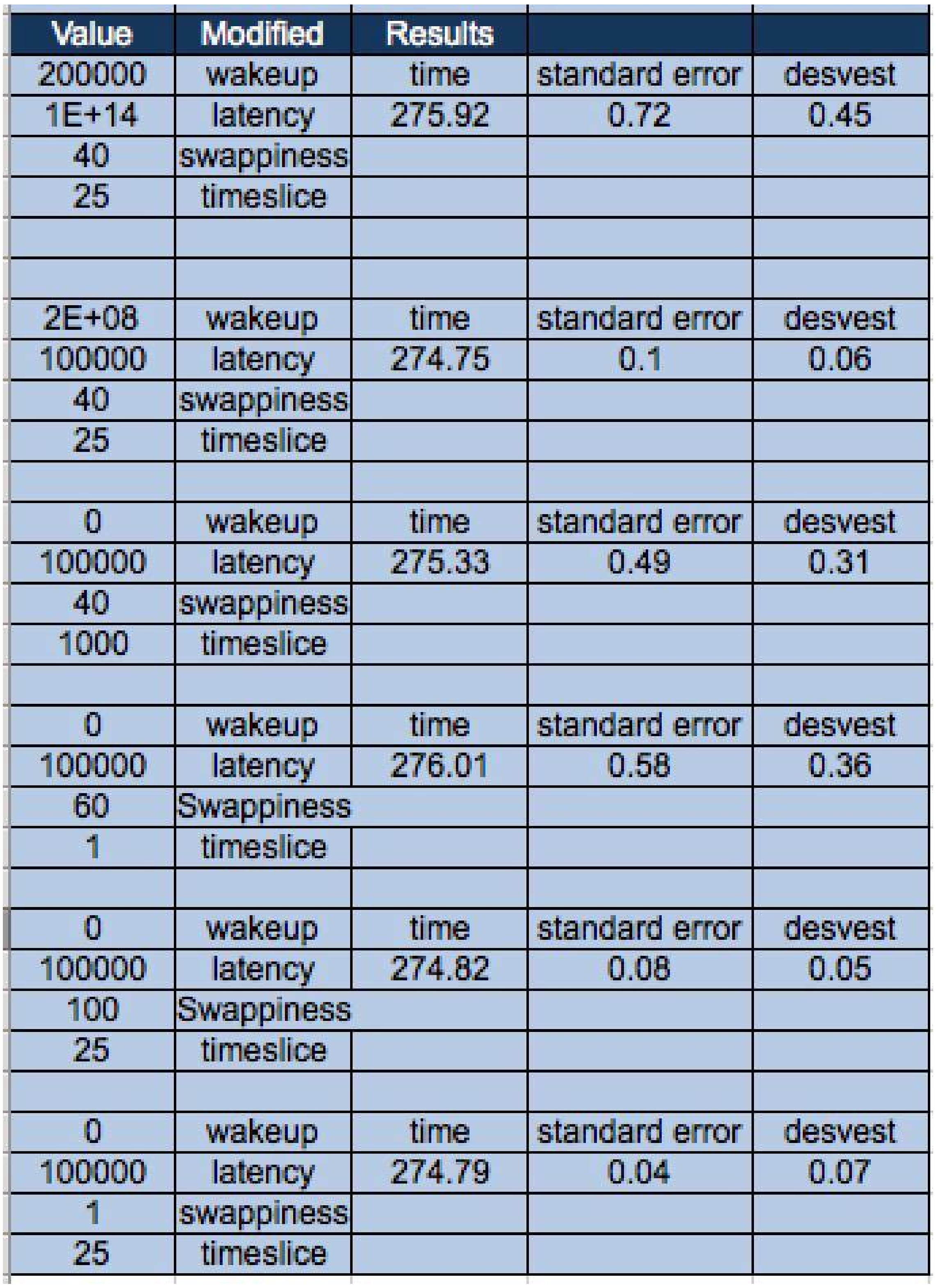
After this case of study we can safely conclude that our theory was wrong and changing the variables, in a computer that utilizes Intel Pentium (R) D CPU 2.80 GHz x 2m, that we picked won't affect the performance of a process such as n-queens.

The n-queens resulted to be a simple process, that is not affected by any of the Kernel components we decided to evaluate. We can refute the initial hypothesis, and we can reassure that any legar number changing latency, timeslice, granularity and swappiness will leave the process the same.



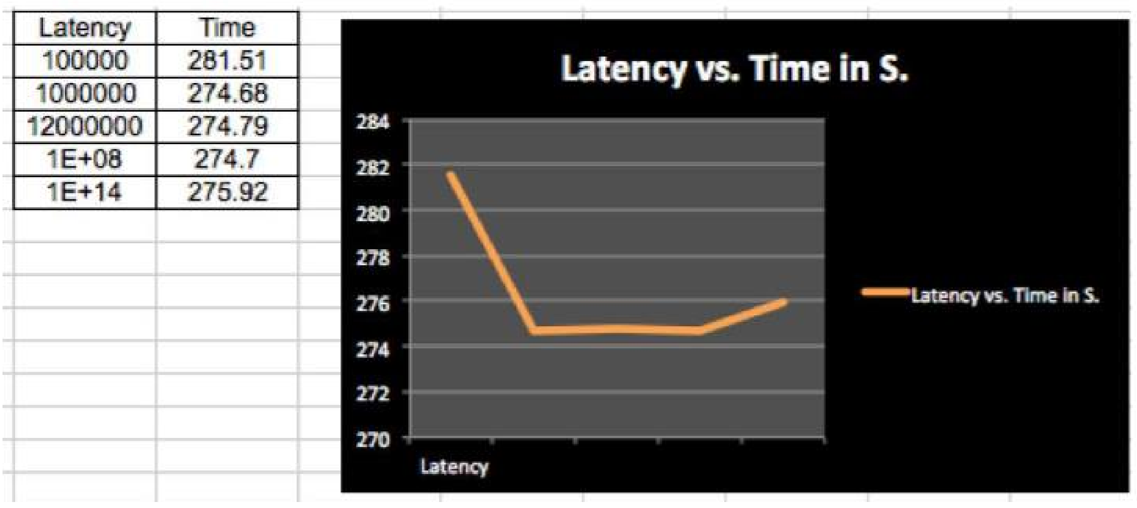
Value	Modified	Results		
2000000	wakeup	time	standard error	desvest
12000000	latency	274.79	0.11	0.07
4000000	wakeup	time	standard error	desvest
12000000	latency	275.56	0.82	0.51
40000000	wakeup	time	standard error	desvest
12000000	latency	274.9	0.11	0.07
200000	wakeup	time	standard error	desvest
12000000	latency	275.53	0.79	0.5
200000	wakeup	time	standard error	desvest
1000000	latency	274.68	0.06	0.04
40	swappiness			
25	timeslice			
200000	wakeup	time	standard error	desvest
100000	latency	281.51	5.21	4.53
40	swappiness			
25	timeslice			
200000	wakeup	time	standard error	desvest
1E+08	latency	274.7	13	0.08
40	swappiness			
25	timeslice			
200000	wakeup	time	standard error	desvest
1E+14	latency	275.92	0.72	0.45
40	swappiness			
25	timeslice			

Figure 1: Modified values.



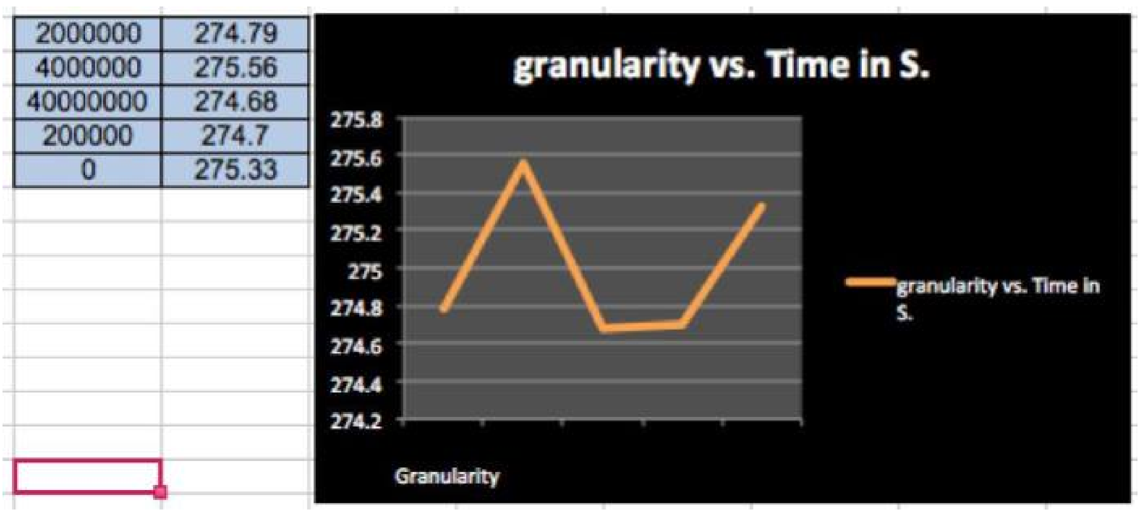
Value	Modified	Results		
200000	wakeup	time	standard error	desvest
1E+14	latency	275.92	0.72	0.45
40	swappiness			
25	timeslice			
2E+08	wakeup	time	standard error	desvest
100000	latency	274.75	0.1	0.06
40	swappiness			
25	timeslice			
0	wakeup	time	standard error	desvest
100000	latency	275.33	0.49	0.31
40	swappiness			
1000	timeslice			
0	wakeup	time	standard error	desvest
100000	latency	276.01	0.58	0.36
60	Swappiness			
1	timeslice			
0	wakeup	time	standard error	desvest
100000	latency	274.82	0.08	0.05
100	Swappiness			
25	timeslice			
0	wakeup	time	standard error	desvest
100000	latency	274.79	0.04	0.07
1	swappiness			
25	timeslice			

Figure 2: Modified values.



Latency	Time
100000	281.51
1000000	274.68
12000000	274.79
1E+08	274.7
1E+14	275.92

Figure 3: Latency vs. Time.



2000000	274.79
4000000	275.56
40000000	274.68
200000	274.7
0	275.33

Figure 4: Granularity vs. Time.

We also found that the standard error changed a lot during the testing progress. sometimes it went up until 6%. Even though, we still have the same average runtimes. the smallest time for the process to be done was 274 seconds, and the time that it took the most it was 281.

Future work could be to look for a different benchmark, one that causes more stress to the computer, so that we can prove if changing those variables will keep the program running at the same speed regardless of the process. Or our future work could be to look for a variable that actually changes the performance of the n-queens.

8 Bibliography

References

[1] Dealy, Sheldon. Common Search Strategies and Heuristics With Respect to the N-Queens Problem. From http://www.cs.unm.edu/~sdealy/nqueens_presentation.pdf

[2] Jacek Kobus and Rafał Szklarski. Completely Fair Scheduler and its tuning. from http://fizyka.umk.pl/~jkob/prace-mag/cfs-tuning.pdf

[3] Silberschatz, A., Galvin, P. B., & Gagne, G. (2005). Operating system concepts. Hoboken, NJ: J. Wiley & Sons.

[4] Sin, Chandandeep. Completely Fair Schedule. Linux Journal. from http://www.linuxjournal.com/magazine/completely-fair-scheduler

[5] Swappiness. from https://sites.google.com/site/tipsandtricksforubuntu/system-tips/swappiness

[6] The Linux / Kernel organization. from https://www.kernel.org/category/about.html

[7] Tips and tricks for Ubuntu. Turning the tasks scheduler. from Chapter 14. Tuning the Task Scheduler