Research Reproduction

The Linux Scheduler: a Decade of Wasted Cores

By William Walcher and Eric Semeniuc

Due September 23rd, 2019

1 Critique

This paper challenges the assertion that schedulers are a solved problem (ie. work conserving) in the systems domain. In the introduction, the following quote is quite notable:

> "And you have to realize that there are not very many things that have aged as well as the scheduler. Which is just another proof that scheduling is easy." - Linus Torvalds

Many people look at scheduling as something we've figured out in systems, even the creator of Linux! By identifying a set of bugs in the Linux scheduler, proposing a set of fixes and useful new tools for debugging, and measuring the differences, the authors debunk this sentiment that 'scheduling is easy/solved'. Their experience writing tools to find and prevent these bugs reinforces their final claim that schedulers are in fact, an ongoing effort.

The authors begin by hypothesizing that the Linux scheduler fails to be work conserving, ie. utilize idle cores when there are cores with run queues containing more than one thread. This violates the invariant of a scheduling algorithm - to always make sure ready threads are scheduled on available cores.

In order to test this hypothesis, the researchers use the following experimental setup:

1. Test hardware:
   (a) **CPUs:** 8 x 8-core Opteron 6272 CPUs (64 threads total)
   (b) **Clock frequency:** 2.1 GHz
   (c) **Caches (per CPU):** 768 KB L1, 16 MB L2, 12 MB L3
   (d) **Memory:** 512 GB of 1.6 GHz DDR3
   (e) **Interconnect:** HyperTransport 3.0
2. Linux kernel: According to the paper, any Kernel from version 3.9 up to 4.3. The patches provided are for the early 4.1.x series.
3. Software to run:
   (a) NAS applications [1] (bt, cg, ep, ft, is, lu, mg, sp, ua)
   (b) TPC-H benchmarks [3]
   (c) R
   (d) GCC (for compiling a kernel)
4. Software tools for system analysis:
   (a) Online sanity checker:

i. Periodically checks for violations of the invariant that idle cores should never be idle when there are busy run queues

(b) Scheduler visualization tool:

i. multiple kernel functions are implemented to record changes to the size of the run queues (<150 loc added)

The researchers ran tests to see how the scheduling algorithm performed, focusing on high intensity scientific computing and multi-threaded applications. These tests included:

- *The Group Imbalance Bug Test:*

  Two R processes are run alongside the compilation of the kernel (make with 64 threads). The scheduler visualization tool is then used to analyze and plot runqueue loads over time.

- *The Scheduling Group Construction Bug Test:*

  The various NAS applications mentioned above are launched on two NUMA nodes with an equal number of threads as there are cores. Execution times are then compared between the buggy and patched versions of the scheduler.

- *The Overload-on-Wakeup Bug Test:*

  TPC-H request #18 and the full TPC-H benchmark are run for both the buggy and patched schedulers. Execution times are then compared between the two.

- *The Missing Scheduling Domains Bug Test:*

  Several NAS applications (as described above) are run with both the buggy and patched version of the scheduler. In the buggy version, a core is first disabled, then re-enabled to produce the bug prior to running the programs. Then, the times are compared between both versions.

The pros of this experimental setup and the tests run include the fact that the software tools (R, TPC-H, NAS) and Linux kernel are readily available online for free, along with how hardware setup being used is laid out - assuming you have access to this hardware. The cons of the experimental setup included a lack of documentation on how the test setup was constructed. The following were notable omissions:

1. What database was used (eg. Oracle, SQL Server, DB2, etc)
2. Was the system I/O limited (ie. HDD vs SSD/NVMe)
3. What OS and distribution was used
4. What exact kernel version was used
5. Did the kernel have extra security patches for current hardware exploits (eg. Spectre, Meltdown)
6. What version of the benchmarks were used (ie. OMP or MPI)
7. What parameters were used on the benchmarks (ie. scale factor)
8. What compiler toolchain generated the code
9. What compilation flags were used (TPC-H has specific flags)

Each of these variables required us to make guesses based on the available options, and hope that they would be compatible with each other. In addition, the highly specific nature of the hardware used could make reproduction of the results more difficult compared to say, some virtual machine image that could be easily distributed.

The results of these experiments were impressive, and we thought that the research was carried out well. There is a succinct hypothesis, experimental setups for each bug are laid out clearly, and

the results seem to back up the claims made by the authors. Each bugfix is backed up by numbers that show a clear improvement for the cases that the researchers threw at the machine. Take for instance, the `lu` NAS command, which shows a 27x speedup post-fix for the group construction bug, and a 137.59x speedup for the missing domains bug. The fact that so many concrete numbers are provided makes it much easier for the reader to believe that the results presented are indeed truthful. Besides just numbers, the visualizations presented of per-core run queues were very useful in both understanding the bugs presented, as well as visualizing the extent of the effect for each fix.

We thought that the cause-effect-fix layout used to approach the introduction and discussion of each bug made the paper much more digest-able. The layout allows readers to go from zero knowledge about the inner-workings of the Linux scheduler, to a deep enough understanding such that the bug explained makes sense. It was also interesting how the authors' experience designing tools to find and prevent these bugs was used to reveal a large complexity in the design of the scheduler; namely due to the ever-increasing complexity of hardware, and the continued attempts by programmers to leverage new hardware functionality. By having to navigate the codebase of the scheduler when crafting their debugging tools, the team concluded that a scheduler should ideally be built in an almost microkernel-esque 'modular' fashion. A core scheduler satisfies the basic invariant of making sure that no core is idle when there is work to be done, and multiple other specialized schedulers can make suggestions to the core scheduler to deal with considerations regarding caching and locality of nodes. This concept of modularizing hardware-dependent software to make more efficient use of the underlying hardware (while not new) is a very intriguing area, one that we think deserves further exploration, especially as systems get more heterogeneous.

Had we had the chance to change things, we would have chosen to use a more portable system for the testing of these bugs - something like a VM/Docker image, which could be easily shared with other people interested in our research. We also would have tried to find a way to use hardware that was more readily available, but that still exhibited the bug. Documentation could have easily been made as well, so that other researchers would be able to simply download our virtual disk images (with all tools/software nicely packaged) and run the tests for themselves to verify our work, on more typical (say, 4 core) computers.

From reading this paper critically, we learned that modern hardware has made the once seemingly simple problem of scheduling processes much more complex. There are now multiple considerations to be made when choosing how to schedule threads in a modern system, that go beyond the simple 'is a core idle' invariant (i.e. cache locality for NUMA groups). The hierarchical structure of the CFS scheduler and the considerations it makes to weights and load were illuminating to thinking about how to effectively utilize many cores. Despite programmers' best efforts to continue to exploit performance from these new systems, they have failed to make a scheduler that performs optimally. A key cause of this failure lies in the fact that modern debugging tools for kernel-level code are still lacking substantially. We found that the ideas presented with respect to better tools were valuable, and something that the systems community as a whole should continue to focus on as systems become more and more complex.

2 Reproduction

For our reproduction, we chose the figure regarding TPC-H performance benchmark runtimes (table 2). To reproduce this figure we used the following setup:

2.1 Hardware

1. **CPU:** Intel Core i5 4670K (4 core)
2. **Clock frequency:** 3.4 GHz

3. **Caches:** 4 x 32KB, L2 4 x 256KB, L3 6MB (shared)
4. **Memory:** 16 GB of 1.6 GHz DDR-3
5. **Disk:** 128 GB USB mSATA SSD

2.2 Software

1. **Distribution:** Alpine Linux 3.3.0 x86_64
2. **Stock Kernel:** 4.1.15-0-grsec
3. **Tested Kernel:** 4.1.52 (vanilla)
4. **Compiler:** GCC 5.3.0
5. **Libc:** musl 1.1.12-r8
6. **Database:** PostgreSQL 9.4.15
7. **TPC-H:** 2.18.0_rc2
8. **QEMU:** 4.1.0

2.3 Test Setup

We began by finding a Linux distribution that ran a kernel susceptible to the bugs presented. We decided on Alpine Linux version 3.3, since it is lightweight and runs Linux kernel version 4.1.39, which (according to the authors) should have the scheduling bugs present.

Next, we used QEMU to boot the disc image, and downloaded the source for the TPC-H benchmark [4]. We also downloaded PostgreSQL, and set up a PostgreSQL database for the queries to run on. With TPC-H and our system up and running, we patched the kernel with the patches provided by the authors of the paper [5]. We now had two disk images - one with the patches applied, and one without. The steps for reproducing the system setup, kernel patches, and installation are available [2].

Unfortunately at this point, things began to fall apart. Despite multiple attempts to speed up QEMU, the benchmarks ran incredibly slow. On top of this, the TPC-H queries kept hanging (possibly due to some lock contention issues, as evidenced by error messages we saw about CPU soft lockups - even running on a single core VM). As a result, we decided to convert our virtual disk images for our patched and unpatched kernels to disk images that could be booted on physical hardware. This was accomplished by using the qemu-img convert tool. The resulting images were then made into bootable drives on an external SSD, and were then booted on the system specified above.

One clear issue we noted was that the hardware we ran the benchmarks on differs greatly from that of the hardware the researchers ran their benchmarks on. In order to account for this, we present the original numbers, as well as a set of numbers obtained from different core counts in an attempt to extrapolate a trend in speedups. It should also be noted that the overload on wakeup bug does not depend on the underlying system architecture (i.e. NUMA groups) to the same extent that the other bugs do. It deals with load balancing issues between per-core run queues, and not NUMA groups. For this reason, we figured it was a better bug to test on our (non-NUMA) hardware.

The following table encapsulates the different times we obtained for both the patched and buggy versions of the kernel, running TPC-H query #18 alone, as well as the entire suite of queries (1-22). It should be noted that we were unable to run query 20, as it never terminated. As a result, it had to be excluded from the full suite benchmark time.

2.4 Test Results

Bug Fixes	TPC-H Query #18	Full TPC-H Benchmark
None	7.21s	4785.77s
Overload-on-Wakeup	7.14s (-1%)	4454.91s (-7%)

The following table shows the runtime of TPC-H for a 2 core machine, with and without the overload on wakeup patch. We obtained a 2 core machine by setting maxcpus to 2 when booting on our machine. The goal of these runs was to try to extrapolate a trend in whether, as core number increased, the bug manifested itself more (i.e. if the relative speedup for 4 cores was greater than 2).

Bug Fixes (2 Cores)	TPC-H Query #18	Full TPC-H Benchmark
None	7.14s	4465.07s
Overload-on-Wakeup	7.12s (-0.3%)	4455.55 (-0.3%)

Looking at the 2 core speedup percentages in comparison to the 4 core percentages, it does indeed seem like there may be a trend towards a higher speedup as cores are added, despite there being insignificant speedups overall on our machine.

3 Critique Addendum

3.1 Differences

Taking the differences in hardware into account, our initial results make sense. The bugs presented in the paper clearly are not a big problem on the average consumer's 4 core computer at the moment. This makes even more sense, as it likely explains why developers of the kernel did not see this bug when they tested on their (likely not 64 core) hardware. Seeing as the machine they used was 64 cores, we can see how they could experience such a significant speedup when they were able to distribute work across the cores. With a small number of cores it seems nearly impossible to see a significant effect, however.

One glaring inconsistency with our results and those published in the paper was immediately clear when we obtained times of $\sim$7 seconds for TPC-H query #18, whereas the paper reports numbers in the 40-50 second range. This was made even more strange by the fact that, even though the researchers claimed query #18 was one of the queries "most influenced by the bug", we saw larger variations in runtime for query #17 between the patched and unpatched kernels. A possible explanation for this difference in runtimes is that we used PostgreSQL for our database, when the researchers were using an undisclosed commercial database.

3.2 Reproduction Challenges

Without a doubt, we can say that this seemingly straightforward reproduction was much more difficult than it should have been. Beyond the somewhat vague documentation of the test setup mentioned above, we ran into numerous issues. The first set of issues we experienced revolves around QEMU, as we needed a way to emulate the hardware used in the paper. VMs do not have the required performance to emulate a 64 core 8 NUMA machine when running on 4 core machines with intensive queries. Even when we tried to emulate the setup in the paper, we ran into multiple CPU lockups that required us to take a different approach. We ran into issues with getting networking set up and configured to work in our QEMU VM. This was needed so we could download dependencies and TPC-H, which sidetracked us for several hours.

The second set of issues revolves around TPC-H, and the lack of clear documentation. Getting TPC-H compiled had some stumbling blocks with manual Makefile configuration. The included DDL generator and query generator are arcane and required combing through internet search results as the documentation for it was lacking. A lack of info in the paper also made some parts needlessly difficult. Additionally, manual hacks and patches were needed to successfully compile the queries on PostgreSQL as the SQL `WHERE` syntax was not conforming.

Once we managed to get the queries running, we ran into a third set of issues with building the kernel. When building kernels in the Alpine distribution, several documented dependencies exist, which appear partway through the compilation process. The Linux kernel requires `coreutils bison flex openssl-dev linux-headers libelf-dev bc perl`, with the exception of linux-headers, none of the dependencies come to mind. For each dependency, the compile failed midway, and manual google searches needed to be conducted. Once all the dependencies were acquired, and assembler errors showed up, to which the internet was of no assistance. This forced us to compile on an Arch Linux system and manually copy the compiled kernel and associated models back to the VM each time.

Having gone through the physical hardware transition, we still encountered issues where queries would stall out, including query 20 which (even when left running overnight multiple times) did not ever terminate on any machines. As a whole, running the tests became very time consuming as they needed constant 'baby-sitting' to bring them to completion.

These challenges in aggregate did not change our assessment of the paper, but they do detract from the overall fit and finish of the paper. For example, uploading a disk image with the test procedures would not be very difficult, nor would explicitly listing out all the test setup details in an appendix.

References

[1] Nas applications. URL https://www.nas.nasa.gov/publications/npb.html.

[2] Os setup steps. URL https://github.com/wjwalcher/LinuxSchedulerReproduction/blob/master/reproduction_steps.txt.

[3] Tpc-h benchmarks, . URL http://www.tpc.org/tpch/.

[4] Tpc-h download, . URL https://github.com/tpc-h/tpc-h.

[5] A decade of wasted cores repo. URL https://github.com/jplozi/wastedcores.