CS736 Project 1 Report: Reproducing the Performance Evolution of Linux Core Operations

Author: Junda Chen
Group Member: Tarun (complete separately and compare result)

Machine Spec

(Machine used in this project) M510 (Utah) cloudlab server
Processor: 8-core Intel(R) Xeon(R) CPU D-1548 @ 2.00GHz
Cache:
- L1 cache: 512 KB
- L2 cache: 2 MB
- L3 cache: 12 MB

Memory: 4x 16 GB DDR4-2133 SO-DIMMs
Storage: 256 GB NVMe flash storage (formatted with ext3 file system)

HP DL160 G9 server (Machine used in the [LEBench] paper)
Processor: 2.40GHz Intel Xeon E5-2630 v3
Cache:
- L1 cache: 512 KB
- L2 cache: 2 MB
- L3 cache: 20 MB

Memory: 128GB of 1866MHz DDR4
Storage: 960GB SSD

Code changes

To make sure no error is produced, I changed the code such that every printf statement that prints a fatal error message (originally prefix [ERROR]) will lead to a program exit. I found a few bugs in the code listed as follows:

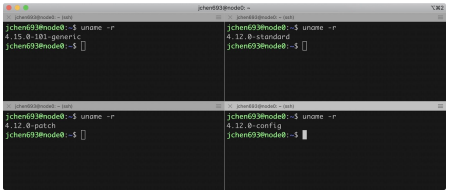
1. **Socket path**. In the send() and recv() test, the authors accidentally wrote `strncat` to `strncpy`, causing the path to overwrite (but not concat). Originally the socket file should write to $LEBENCH_HOME/TEST_DIR/socket, yet the bug caused the socket to open at TEST_DIR/socket, a non-existing directory that silently slipped away. I also made a pull request to the original code base that has proposed to change this behaviour.
2. **Socket accessibility**. In some unknown rare cases, a program may not be able to access the previous socket file, causing the program to silently fail. I add remove the socket if that socket is not accessible during the test. I followed Piazza comment @27_f2 to fix the bug.
3. **Uninitialized values**. Uninitialized values can cause the socket connection to fail. I added the initialization of the values as stated in Piazza comment @35_f1.
4. **Rename csv file.** I add a timestamp at the output csv file to make sure runs on the same kernel name do not overwrite each other.

Test runtime is relatively acceptable (around 20-50 mins per round).

OS Version and Test Setup

I used the following version for each tasks:

- Task1: Linux 4.15, the standard version provided by the cloudlab machine.
- Task2: Linux 4.12
- Task3: Linux 4.12, with a patch on the fault around
- Task4: Linux 4.12, unset config `CONFIG_MEM_CG`



The experiment setup is stated as follows:

1. Allocate bare metal machines from cloudlab.
2. Download linux-stable and compile images with different config / patch.
3. Download LEBench and debug.
4. Ensure a smaller test suite success (OS_Eval.c, with small iterations for each test)
5. Setup the kern_list and initial cron job.
6. Run the test.

Absolute Performance Result

Figure 1 shows the absolute test latency among the 4 OS versions on each of the tests provided by LEBench. LEBench has 24 metrics shown in their paper, each metric can be associated with different tests in their code[1]. **Appendix 1** lists the absolute performance results, in numeric table, among the 4 different OS versions.

For comparison, **Figure 2** shows the heat-map-style test latency comparison similar to the heatmap in original paper (figure 1). **Figure 2(a)** replicates the heat-map in original paper, showing the k-best result only for 4.12 and 4.15. **Figure 2(b)** shows the k-best solution from my experiment, and **figure 2(c)** shows the *average performance*. The k-best solutions are taken for k=5%. Notice that the two figures have different baseline to compare with (2a is based on 4.0, and 2b is based on 4.12), so a direct numeric difference cannot be interpreted. But overall, the three graphs show not much difference in their color code separation.

Analysis: Linux 4.15 (default) vs Linux 4.12 (default)

Not surprisingly, 4.15 (default) performs worst or equally bad among all OS versions. In all tests except for big send, fork and big munmap, 4.15 had the longest latency among all.

Reviewing the paper, we may correspond the slowdown of small read and recv to the *kernel page table isolation* introduced in Linux 4.14 ([LEBench] Section 4.1.1), and the slowdown of poll and recv probably to *indirect branch speculation* in Linux 4.14 ([LEBench] Section 4.1.2). Sadly, I did not patch these on our Linux 4.15 image (see lesson learned), and I cannot assert these are the root cause of the performance degrade in Linux 4.15 compared to Linux 4.12.

Surprisingly, the getpid() test on 4.15 standard ran 570% slower than on the 4.12 standard version. I do not understand why this behaviour exists, and the paper does not seem to document this either.

[1] For example, the entry *"send & recv"* is associated with 4 different test cases in the figure: send, recv, big send, big recv.

Analysis: Linux 4.12 (patch, fault-around) vs Linux 4.12 (default)

The Linux 4.12-patch (3rd column) version adds a patch to the fault-around. As expected, it runs fastest on page fault tests because it reduces overhead on a page fault. Compared to the default patch, the only noticeable performance gain are also in the page-fault tests.

Analysis: Linux 4.12 (config, `MEM_CG`) vs Linux 4.12 (default)

The Linux 4.12-config (4th column) version unset the `MEM_CG` config variable to disallow the cgroup controller from synchronously uncharging resource usage, which can negatively hurt performance. As described in the [LEBench] paper, this version does give a slightly better performance gain (-10% latency) on munmap tests.

Unlike what is described in the paper, the big/huge page fault tests observe a -15% performance gain in latency, and also a roughly -10% gain in big send / recv. In addition, tests such as fork and mmap also observe noticeable performance gain above -5% percent.

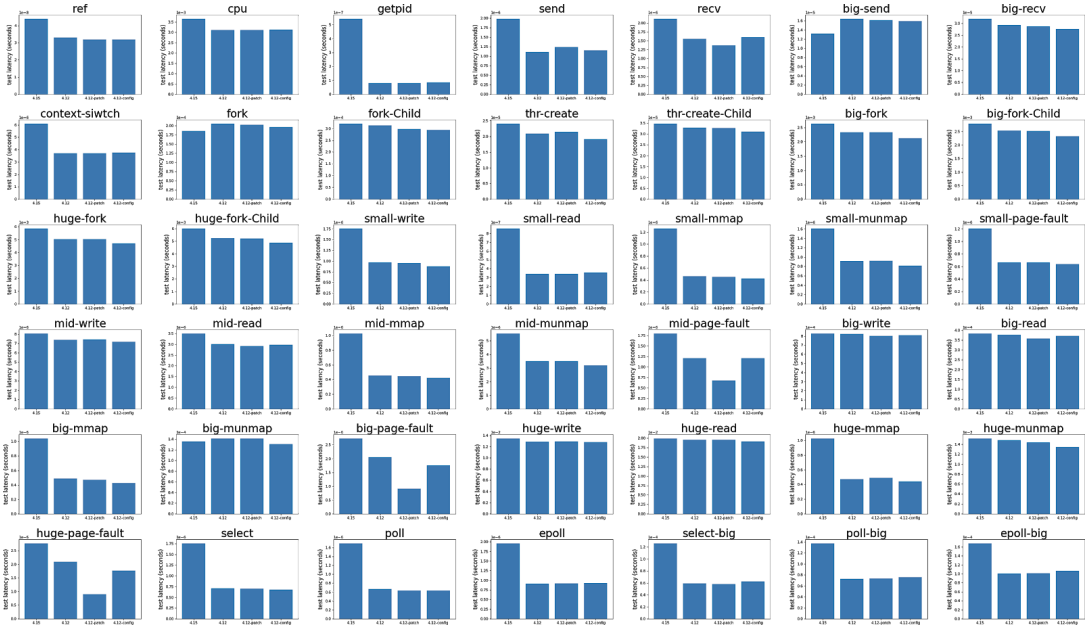


Figure 1. Absolute performance comparison between OS Versions. The OS versions in the four bars from left to right are: 4.15 (default), 4.12 (default), 4.12 (patch, fault–around), 4.12 (config, MEMCG). We took the average value range from 10 different test runs.

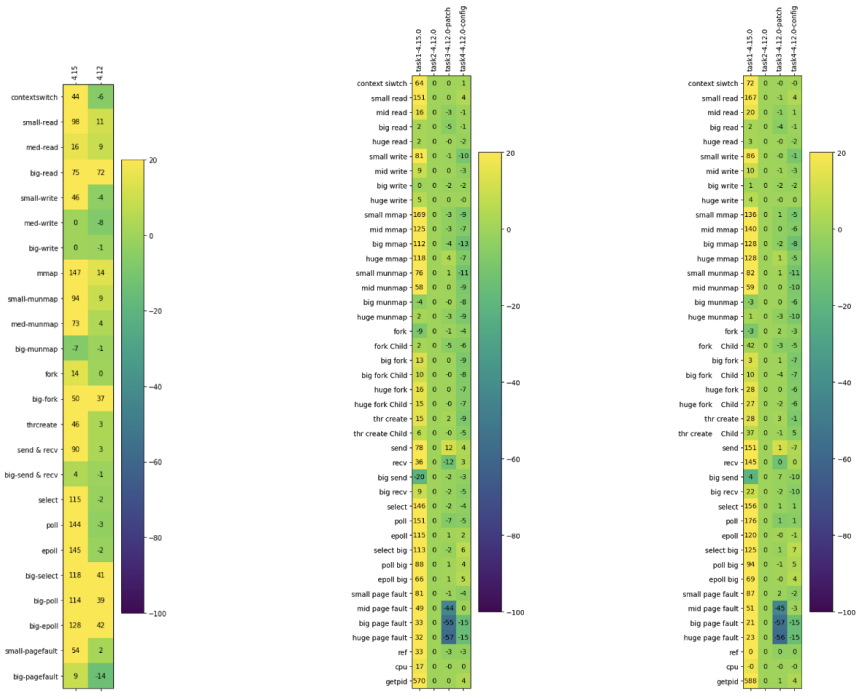


Figure 2. (a) The relative performance (k–best) between 4.12 and 4.15 (baseline 4.0, not shown here) in the paper. (b) My result (k–best) between the four variation of OS (baseline 4.12–standard, the 2nd column) (c) My result (average) between the four variation of OS (baseline 4.12–standard, the 2nd column)

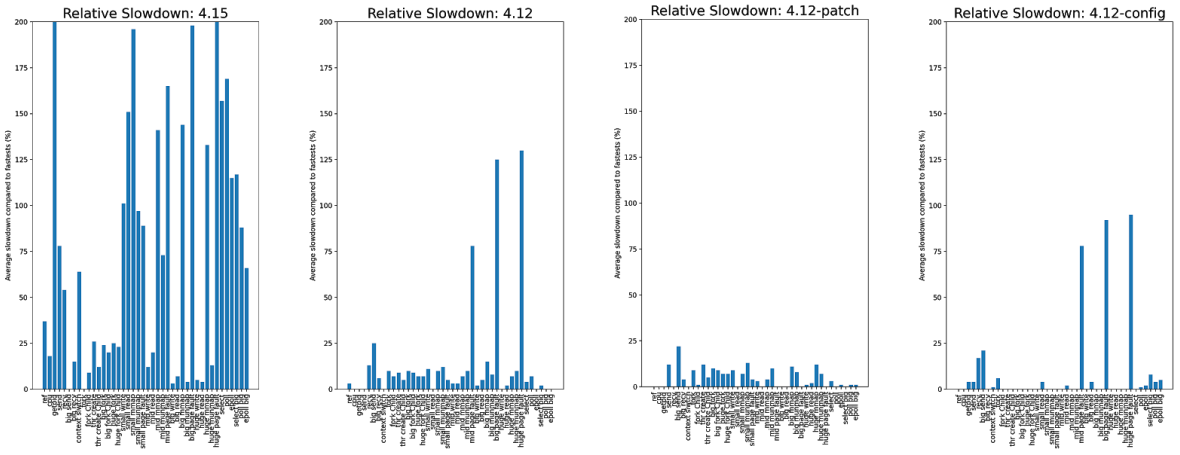


Figure 3. Relative slowdown. This shows the relative slowdown (percentage) of each test compared to the fastest test among all. Y–axis value trimed at 200 for readability.

Relative performance results

Figure 3 shows the relative performance result. As expected, 4.15 (default, top left) remains the worst in most metrics. 4.12-patch (bottom left) fixes the fault-round and therefore gets the best performance in page-fault related tests. Similarly, 4.12-config (bottom right) unset the `MEM_CG` and become the fastest in `munmap` tests.

Comparison to [LEBench] paper

As mentioned in the previous two sections of analysis, the result shown here overall matches the result shown in the paper, with some differences in performance metrics that are hard to compare and verity.

Compared to the patched version, the performance of page-fault in (b) (d) shows around 100-125% slowdown. This is partially aligned with the statement in the table3 of the paper that disabling fault-around can get around 165% performance gain.

In addition, (d) runs fastest in fork- and thread-related tests. Other versions are generally 10-20% slower. This fact does not seem to be reflected in the [LEBench] apper.

Lesson Learned

Allocate the same type of machine for testing.
When allocating machines from cloudlab, I have ignored the option “machine type” and allocated different types of machines for the tests. I unfortunately found out two days before my due day, in which at the same day all the Wisconsin nodes with type c220g1 have been allocated. As a result, I had to run all the tests again on a m510 Utah cluster.

Test on Linux 4.0 for baseline.
Figure 2 (a) and (b) cannot be directly compared with each other. We have to know the 4.0 base performance in order to do so. Therefore, knowing the performance on 4.0 really should have been a part of the project.

I regret that I realized this too late at the analysis phase, and I am surprised that no one brought this up on Piazza as a question. I think the authors should at least provide the absolute performance (in seconds) on the Linux 4.0 machine to make this work “reproducibly comparable” (maybe a bit harsh though…).

Pick Linux 4.15 for task3 (patch) and task4 (config) instead of Linux 4.12.
I picked Linux 4.12 for task3 and task4 as I want to make the Linux versions consistent. I should have picked Linux 4.15 and show how much performance is gained compared to Linux 4.12. With the current setup, we don’t really see a drastic difference among versions - and it’s definitely less exciting to do so.

Appendix

- Appendix 1: Absolute Performance Comparison (screenshot attached below)

Appendix 1 Screenshot: Absolute Performance Comparison between OS Versions

Absolute performance Result

		ref kbest	ref average	cpu kbest	cpu average	getpid kbest	getpid average	send kbest	send average	recv kbest	recv average	big send kbest	big send average	big recv kbest	big recv average
1	4.15.0-101-generic	2.80E-08	4.40E-08	3.08E-03	3.65E-03	5.30E-07	5.43E-07	8.92E-07	1.99E-06	1.18E-06	2.12E-06	1.09E-05	1.32E-05	2.02E-05	3.20E-05
2	4.12.0-standard	2.80E-08	3.30E-08	3.08E-03	3.12E-03	7.70E-08	8.10E-08	3.55E-07	1.11E-06	4.81E-07	1.56E-06	1.05E-05	1.65E-05	1.66E-05	2.93E-05
3	4.12.0-patch	2.80E-08	3.20E-08	3.08E-03	3.11E-03	7.80E-08	8.10E-08	3.59E-07	1.24E-06	4.83E-07	1.37E-06	1.12E-05	1.62E-05	1.62E-05	2.88E-05
4	4.12.0-config	2.80E-08	3.20E-08	3.08E-03	3.12E-03	8.00E-08	8.40E-08	3.29E-07	1.16E-06	4.81E-07	1.60E-06	9.45E-06	1.59E-05	1.49E-05	2.77E-05

		context siwtch kbest	context siwtch average	fork kbest	fork average	fork Child kbest	fork Child average	thr create kbest	thr create average	thr create Child kbest	thr create Child average	big fork kbest	big fork average	big fork Child kbest	big fork Child average
1	4.15.0-101-generic	5.00E-06	6.10E-06	8.13E-05	1.86E-04	1.74E-04	3.21E-04	1.05E-05	2.41E-05	1.56E-05	3.47E-05	1.82E-03	2.63E-03	1.99E-03	2.79E-03
2	4.12.0-standard	2.90E-06	3.72E-06	8.39E-05	2.05E-04	1.23E-04	3.14E-04	8.22E-06	2.10E-05	1.14E-05	3.28E-05	1.76E-03	2.33E-03	1.81E-03	2.54E-03
3	4.12.0-patch	2.89E-06	3.72E-06	8.52E-05	2.02E-04	1.19E-04	2.98E-04	8.47E-06	2.15E-05	1.12E-05	3.28E-05	1.78E-03	2.33E-03	1.74E-03	2.53E-03
4	4.12.0-config	2.90E-06	3.76E-06	8.17E-05	1.96E-04	1.17E-04	2.94E-04	8.15E-06	1.92E-05	1.19E-05	3.11E-05	1.63E-03	2.13E-03	1.69E-03	2.32E-03

		huge fork kbest	huge fork average	huge fork Child kbest	huge fork Child average	small write kbest	small write average	small read kbest	small read average	small mmap kbest	small mmap average	small munmap kbest	small munmap average	small page fault kbest	small page fault average
1	4.15.0-101-generic	4.64E-03	5.86E-03	4.79E-03	6.00E-03	1.49E-06	1.76E-06	8.15E-07	8.55E-07	9.95E-07	1.26E-06	1.57E-06	1.61E-06	1.16E-06	1.21E-06
2	4.12.0-standard	3.63E-03	5.03E-03	3.78E-03	5.23E-03	7.97E-07	9.69E-07	3.05E-07	3.41E-07	4.21E-07	4.68E-07	8.62E-07	9.12E-07	6.20E-07	6.68E-07
3	4.12.0-patch	3.65E-03	5.04E-03	3.72E-03	5.21E-03	7.95E-07	9.55E-07	3.03E-07	3.41E-07	4.25E-07	4.56E-07	8.68E-07	9.21E-07	6.32E-07	6.64E-07
4	4.12.0-config	3.41E-03	4.70E-03	3.55E-03	4.89E-03	7.86E-07	8.75E-07	3.18E-07	3.56E-07	4.01E-07	4.26E-07	7.64E-07	8.14E-07	6.05E-07	6.39E-07

		mid write kbest	mid write average	mid read kbest	mid read average	mid mmap kbest	mid mmap average	mid munmap kbest	mid munmap average	mid page fault kbest	mid page fault average	big write kbest	big write average	big read kbest	big read average
1	4.15.0-101-generic	7.84E-06	8.06E-06	3.30E-06	3.50E-06	9.78E-07	1.03E-06	5.35E-06	5.54E-06	1.76E-06	1.80E-06	8.08E-04	8.28E-04	3.76E-04	3.84E-04
2	4.12.0-standard	7.11E-06	7.40E-06	2.74E-06	3.02E-06	4.07E-07	4.57E-07	3.36E-06	3.51E-06	1.16E-06	1.21E-06	7.99E-04	8.24E-04	3.67E-04	3.78E-04
3	4.12.0-patch	7.05E-06	7.42E-06	2.71E-06	2.93E-06	4.08E-07	4.45E-07	3.36E-06	3.52E-06	6.35E-07	6.79E-07	7.81E-04	8.04E-04	3.54E-04	3.59E-04
4	4.12.0-config	6.91E-06	7.17E-06	2.75E-06	2.98E-06	3.83E-07	4.26E-07	3.03E-06	3.20E-06	1.13E-06	1.21E-06	7.83E-04	8.08E-04	3.64E-04	3.72E-04

		big mmap kbest	big mmap average	big munmap kbest	big munmap average	big page fault kbest	big page fault average	huge write kbest	huge write average	huge read kbest	huge read average	huge mmap kbest	huge mmap average	huge munmap kbest	huge munmap average
1	4.15.0-101-generic	1.01E-06	1.04E-06	1.32E-04	1.36E-04	2.39E-06	2.73E-06	1.32E-02	1.35E-02	1.97E-02	2.00E-02	9.88E-07	1.03E-06	1.48E-03	1.52E-03
2	4.12.0-standard	4.41E-07	4.91E-07	1.37E-04	1.41E-04	1.98E-06	2.06E-06	1.27E-02	1.29E-02	1.91E-02	1.96E-02	4.34E-07	4.72E-07	1.46E-03	1.48E-03
3	4.12.0-patch	4.34E-07	4.72E-07	1.37E-04	1.41E-04	8.57E-07	9.16E-07	1.26E-02	1.29E-02	1.91E-02	1.96E-02	4.39E-07	4.91E-07	1.42E-03	1.44E-03
4	4.12.0-config	4.07E-07	4.27E-07	1.28E-04	1.31E-04	1.67E-06	1.76E-06	1.27E-02	1.28E-02	1.87E-02	1.92E-02	4.11E-07	4.40E-07	1.32E-03	1.35E-03

		huge page fault kbest	huge page fault average	select kbest	select average	poll kbest	poll average	epoll kbest	epoll average	select big kbest	select big average	poll big kbest	poll big average	epoll big kbest	epoll big average
1	4.15.0-101-generic	2.42E-06	2.77E-06	1.58E-06	1.76E-06	1.54E-06	1.70E-06	1.73E-06	1.96E-06	1.15E-04	1.26E-04	1.26E-04	1.37E-04	1.49E-04	1.68E-04
2	4.12.0-standard	1.97E-06	2.09E-06	6.18E-07	7.14E-07	5.57E-07	6.74E-07	7.85E-07	9.11E-07	5.11E-05	5.92E-05	6.48E-05	7.31E-05	8.82E-05	1.01E-04
3	4.12.0-patch	8.68E-07	9.07E-07	6.23E-07	7.02E-07	5.62E-07	6.30E-07	7.82E-07	9.18E-07	5.15E-05	5.82E-05	6.44E-05	7.37E-05	8.82E-05	1.01E-04
4	4.12.0-config	1.69E-06	1.77E-06	6.22E-07	6.84E-07	5.60E-07	6.37E-07	7.79E-07	9.31E-07	5.44E-05	6.27E-05	6.78E-05	7.60E-05	9.17E-05	1.06E-04