COMPARISON of SCALABILITY ACROSS DIFFERENT OPERATING SYSTEM KERNELS

by Liang Luo (liangluo), Amrita Mazumdar (amrita), Yuchen Jin (yuchenj)

ABSTRACT

Seven years after the publication of the Linux Scalability paper, we reran a series of multi-core benchmarks, including MOSBENCH, PHOENIX and PARSEC on a similarly configured, same generation 48-Core server and a modern 64-core server with three different kernels (Linux Kernel 2.6.32, Linux Kernel 4.10.1, and Windows Kernel 10.0.14393 for reference). We evaluate overall scalability measured in per-core speedup and kernel scalability based on runtime breakdowns of system and user time. Unfortunately, we did not find meaningful scalability improvement in different generations of kernels, even when testing on new hardware.

EXPERIMENT SETUPS

We ran our benchmarks on Sampa's n02 node, a 48-core (4 x 12-core Opteron 6168) machine with 512GB of memory, and a modern EC2 instance with 64-core (Intel Xeon E5-2686v4) processors. The former processor is roughly the same generation as the processor used in the paper, and the latter gives us a hint on how things may change on a recent core architecture.

We first determined what operating systems could launch what kernel. This table summarizes our process, where a √ represents bootable, - not tried, and × unbootable.

Kernel / OS	CentOS 6	RHEL 6	CentOS 7	Ubuntu 10.04	Ubuntu 16.04	Windows 10 w/ Linux Subsystem
2.6.32	√	√	×	√	×	-
2.6.35.rc5.guk[1]	-	×	-	×	×	-
4.10.1	×	×	√	-	√	-
10.0.14393[2]	-	-	-	-	-	√

We booted two images from NFS with PXEBoot[3] on n02, one containing RHEL 6 with 2.6.32, and one containing CentOS 7 with kernel upgraded to latest 4.10.1. We booted RHEL 6 with 2.6.32 and 4.10.1 kernels on EC2, and a Windows 10 on the same machine.

[1] This is the modified PK kernel. We regret not being able to launch it in on n02. We tried NFS boot, with NFS as built-in or as module, and confirmed the RAMFS contained the module, but during boot the kernel panics with "nfs root type nfs requested but kernel/intrid does not support nfs" or "no root device nfs found". We also tried booting from local disk, resulting in kernel panic "Don't know how to handle /dev/sdc1" and "no root device block/dev/sdc1 found". We tried booting it with REHL 6 and CentOS6; both failed. We also tried to launch it on the cloud, but the VMs we tried (Ubuntu 10.04, RHEL 6) on EC2 could not boot into this kernel either. We eventually gave up after spending 30+ hours trying to boot it.
[2] This is the recent Windows Kernel. We felt it interesting to see the scalability of Windows Kernel by leveraging its Linux Subsystem. We made no change the benchmark and compiled with Ubuntu on Windows.
[3] We also tried booting from virtualization software, but VirtualBox on n02 only booted up to 24 cores, VMWare Workstation up to 16 cores, and we couldn't get KVM to run or Hyper-V to install.

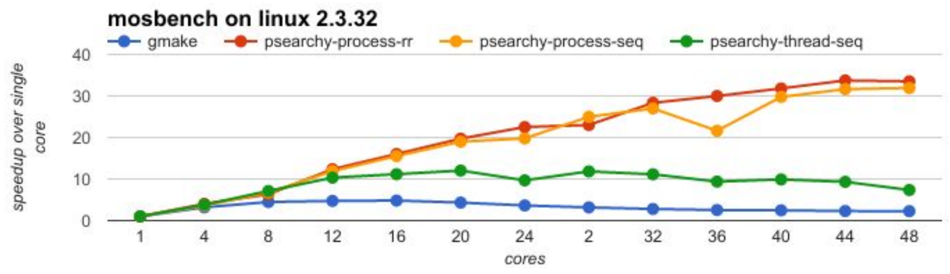
BENCHMARKS

We wanted to benchmark the scalability of the Linux kernels on three benchmark suites: MOSBENCH, the original benchmark suite from the paper; PHOENIX, a MapReduce-based shared-memory implementation of data-intensive processing tasks for multicore/shared-memory processors; and PARSEC, a benchmark suite of multi-threaded programs with heavy use of synchronization. PHOENIX and PARSEC are compute-heavy, and would demonstrate if kernel scalability was a limiting factor for multicore computation tasks.

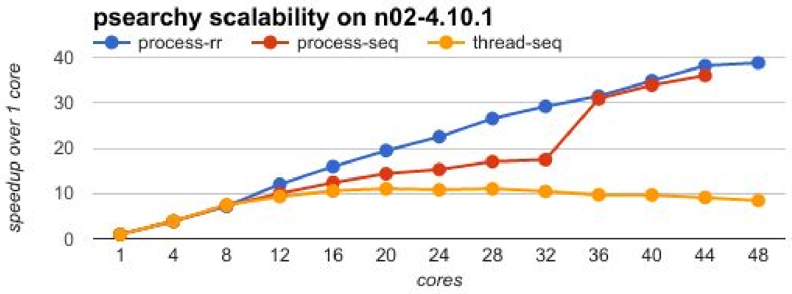
For each benchmark suite, we ran them on our test kernels, varying the number of cores used from 1-48. Varying the number cores used required slight modifications to PHOENIX ; PARSEC included flags to set core count. For benchmarks with very short runtime (i.e. PHOENIX), we inflated the runtime by looping the program to get a better sense of its scaling. In all plots, speedup is measured using wall clock time.

Because of time constraints and technical issues with compiling for different kernels, we could not include all benchmarks. Specifically, we only evaluated `psearchy` and `gmake` from MOSBENCH, `blackscholes`, `dedup`, and `streamcluster` from PARSEC, and `kmeans`, `linear-regression`, `matrix-multiply`, `pca`, `stringmatch`, and `wordcount` from PHOENIX. We measured the runtime of all the benchmarks and evaluated the speedup of multi-core over single-core.

MOSBENCH RESULTS



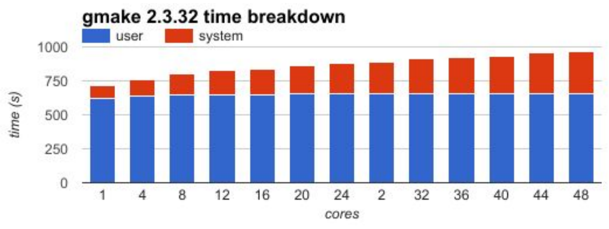
The two MOSBENCH apps we could successfully run were `gmake` and `psearchy`. To reproduce the paper's results, we used the same workloads, and the in-memory file system `tmpfs` in order to avoid disk bottlenecks. We found that `gmake` compiled Linux 2.6.35-rc5 on 2.6.32, but we could not compile it on 4.10.1; `psearchy` indexes the Linux kernel 2.6.35-rc.



We evaluated three different modes of `psearchy`. Thread-seq is the initial version of `pedsort`, which uses a single process with one thread per core. It scales badly since the contention over the shared address space. Process-seq is modified to use one process per core to avoid this scalability problem. However, the active cores spread over few sockets when the number of cores is small.

Process-rr shows performance when the active cores are spread evenly over sockets, which scales better since each new socket provides access to more total L3 cache space .

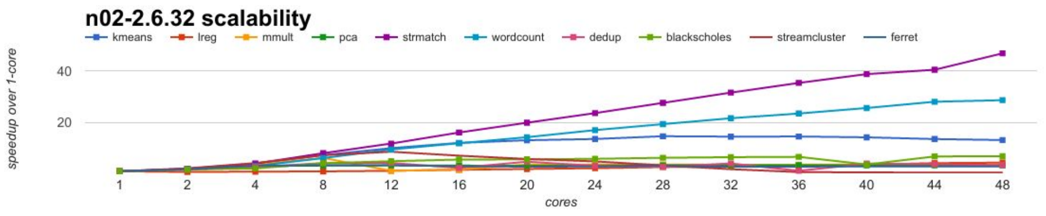
We also evaluated `psearchy` and `gmake` on linux 2.3.2 on n02. We find that `psearchy` has `expected` scaling (~35x) but `gmake` does not scale well at all, achieving a maximum 4.4x speedup.

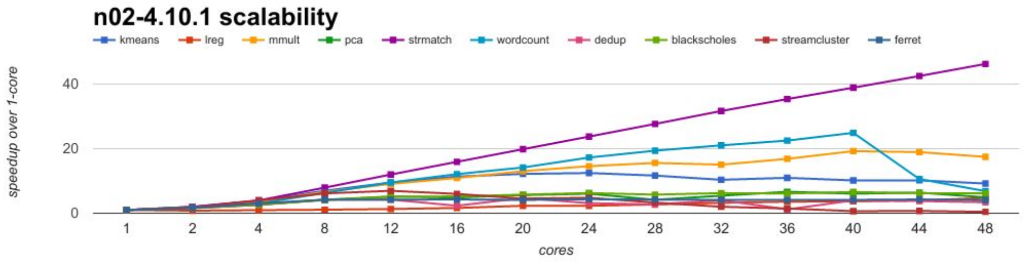


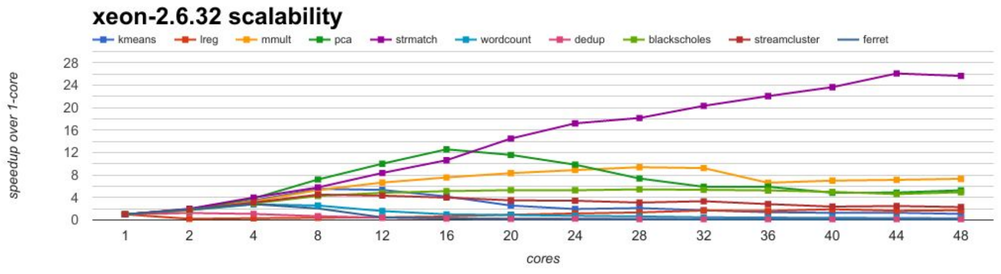
Looking at a breakdown of the time spent in `gmake`, it seems that the time spent doing computation is constant regardless of the number of cores, but the system time grows with the number of cores. This is logical, considering the paper's argument that stock Linux presented scalability obstacles, but not really in keeping with their results, which claimed `gmake` scaled well, running 35x faster on 48 cores with the stock kernel.

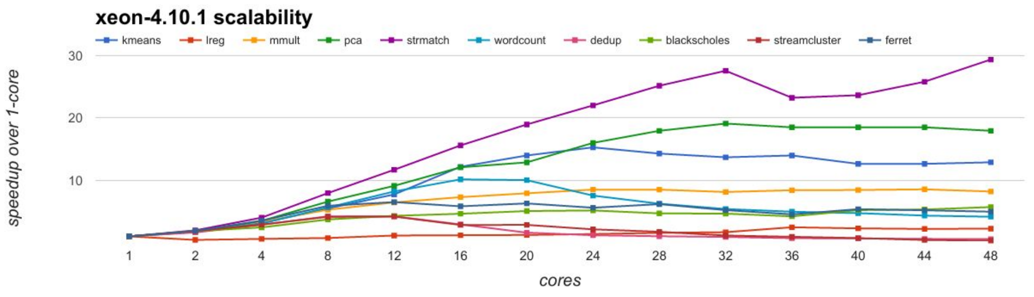
We could not evaluate all of MOSBENCH: 1) Some of the apps (`memcached`, `Apache`, and `Postgres`) require a set of load generating client machines on the same switch, which is infeasible in our cluster environment; 2) `metis` would not compile due to an unknown error; 3) the benchmark `Exim` would appear frozen after the server listens to a port.

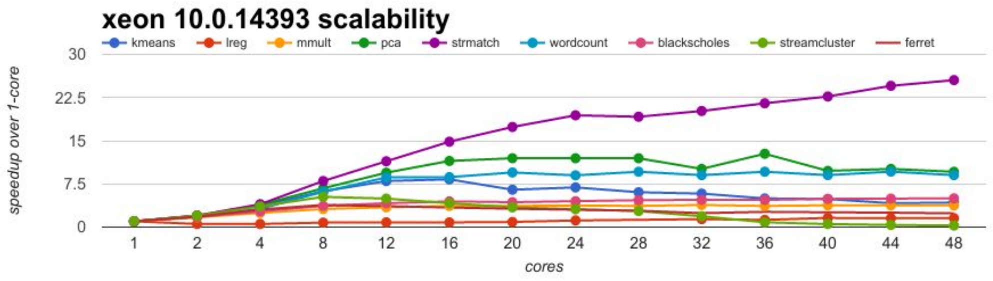
PHOENIX+PARSEC RESULTS







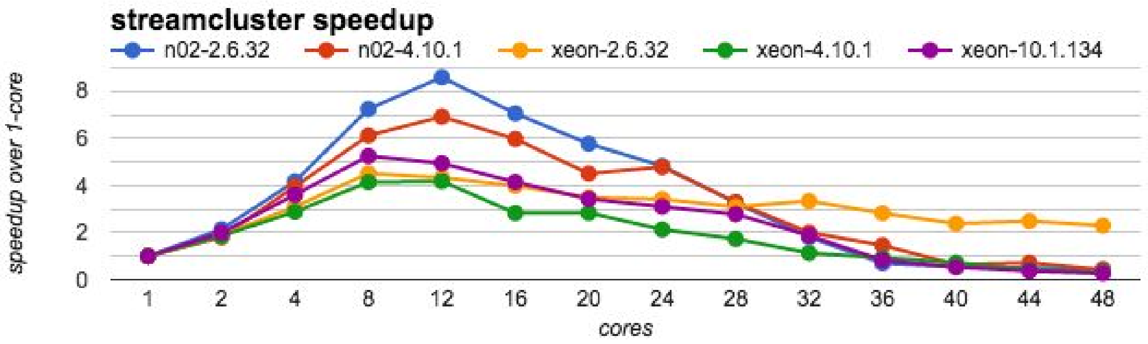




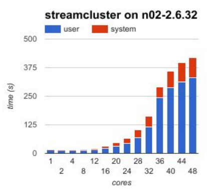
We ideally expect that the speedup will scale linearly with the number of cores in PHOENIX and PARSEC. Some benchmarks present almost-linear scaling of speedup (`strmatch`, and occasionally `blackscholes`, and `matrix-mult`), but most exhibit severely sub-linear speedup. On n02-2.6.32, the most representative configuration of the paper's evaluation, `streamcluster` runs 50% slower at 48 cores than it does with 1 core, and six of the benchmarks (`lreg, mmult, pca, dedup, streamcluster`, and `ferret`) do not achieve more than 8x speedup.

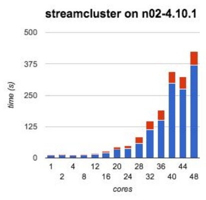
STREAMCLUSTER SCALABILITY

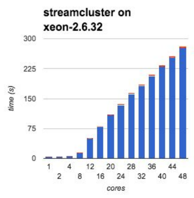
To investigate the speedup results, we examined one kernel in depth, Streamcluster. Streamcluster is a PARSEC benchmark for data mining that clusters data-points to the nearest of a predetermined number of medians. It showed very different scalability on different machines and different configurations, so we dove deeper into the results. The graphs below summarize the results.

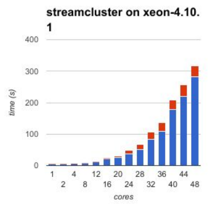


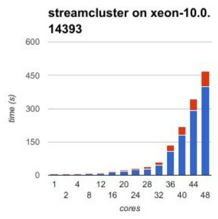
Streamcluster scales with the number of cores up to 8-12, and then falls off sharply, ending in negative scaling for some configurations from 36-48 cores. On both the Xeon and n02, the 2.6.32 scaled better than 4.10.1, with 10.1.134 doing a little better than 4.10.1 on the Xeon.











For streamcluster, the breakdown of time spent in user and system varies across configurations. On n02 running 2.6.32, the time spent in user space grows dramatically at 36 cores, and the time spent in the system does as well. On Xeon running 2.6.32, it looks like the time spent doing computation grows linearly with the number of cores, with little time spent in the kernel. We suspect this sharp increase in user time in all configurations is due to the well-known problem of false sharing, which gets exacerbated with many core, as reported in `[Predator, LASER]`.

CONCLUSION

Our results confirm the paper's claim that Linux presents scalability obstacles, while occasionally not agreeing with the paper's results. We hoped to see an improvement in the scalability of modern kernels, but unfortunately found negligible difference in scalability between different generations of kernels. This indicates the goals of the paper are still relevant in modern kernels, and that the optimizations presented were unexercised across our selection of benchmarks.