Appears in: ACM SIGOPS Operating Systems Review, Vol. 47, Issue 2, pp. 66-72, July 2013.

Our Troubles with Linux Kernel Upgrades and Why You Should Care *

Ashif S. Harji Peter A. Buhr Tim Brecht
Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
asharji,pabuhr,brecht@uwaterloo.ca

Abstract

Linux and other open-source Unix variants (and their distributors) provide researchers with full-fledged operating systems that are widely used. However, due to their complexity and rapid development, care should be exercised when using these operating systems for performance experiments, especially in systems research. In particular, the size and continual evolution of the Linux code-base makes it difficult to understand, and as a result, decipher and explain the reasons for performance improvements. In addition, the rapid kernel development cycle means that experimental results can be viewed as out of date, or meaningless, very quickly. We demonstrate that this viewpoint is incorrect because kernel changes can and have introduced both bugs and performance degradations.

This paper describes some of our experiences using Linux and FreeBSD as platforms for conducting performance evaluations and some performance regressions we have found. Our results show, these performance regressions can be serious (e.g., repeating identical experiments results in large variability in results) and long lived despite having a large negative effect on performance (one problem was present for more than 3 years). Based on these experiences, we argue: it is sometimes reasonable to use an older kernel version, experimental results need careful analysis to explain why a performance effect occurs, and publishing papers validating prior research is essential.

1. INTRODUCTION

Linux and BSD variants currently occupy niche markets, such as supercomputing (94%, Linux) [24], mainframes (70%-80%) [21, p. 3], enterprise servers (42%) [15], and research, versus the desktop market where Windows dominates. These open-source OSs are a boon to academic systems-researchers because they provide a platform to make improvements to and evaluate the performance of systems that are used in production. Prior to these open-source projects, operating system (OS), networking and database researchers were often at the mercy of OS vendors with respect to developing and evaluating new mechanisms or policies. Furthermore, researchers could only find bugs or performance problems in a vendor's OS by treating it as a black box and developing external tests. As well, vendors were frequently unreceptive to performance problems and bug reports. In fact, the availability of open-source OSs has forced some vendors to make some or all their software open source, allowing researchers alternative venues for development. Fundamentally, without access to source code, it is extremely difficult for researchers to innovate in the crucial and expanding area of systems software. Jockeying for special or restricted access to an OS, or working with the OS as a black-box does not allow all researchers equal or sufficient access to find, understand and fix logic or performance problems.

Linux is a popular choice for OS research; however, its main drawbacks are its complexity and rapid development. The Linux kernel has grown to over 15 million lines of code, and on average, a new kernel version is released every 2–3 months with each release currently containing 8,000–12,000 patches [5]. The complexity of a large software system makes it difficult to configure and tune for the best possible performance, and it also makes understanding and explaining the results difficult. In addition, the rapid kernel development cycle means that experimental results can be viewed as out of date, or meaningless, very quickly. But most importantly, the rapid changes in kernel development introduce both bugs and performance degradations. While bugs causing failures are quickly identified and fixed, performance related problems are extremely difficult to isolate and correct. Furthermore, because of conflicting goals and tradeoffs that are central to systems implementation, changes that increase performance in one area may degrade performance in another.

In this paper, we demonstrate significant performance problems exist across multiple Linux and two FreeBSD kernels, and as a result:

- We contend that to encourage good science, publishing papers that validate prior research results is essential.
- We argue that experimental results need careful analysis to understand and explain why a change has or has not produced a performance effect.

*This paper is an extended version of our 2011 APSys workshop paper [9].

- We explain why finding and fixing performance problems is difficult and time consuming, as is getting performance fixes into the Linux kernel.
- We describe why changing to the newest kernel is neither a panacea nor a requirement for sound research.

Finally, we make a number of recommendations for performing sound experimental performance evaluations. These recommendations reinforce an on-going effort in Computer Science and other areas of Science to strengthen the quality of experimental work [1, 2, 6, 7, 12, 14, 18, 23].

2. EXPERIENCES WITH BUGS

We conduct research into designing and testing web-server architectures on uniprocessor and multiprocessor hardware with the goal of understanding how differences in server software architecture affect performance. A *web-server* is the software component through which most Internet traffic flows, hence it must provide high throughput while supporting a large number of concurrent connections. During detailed comparisons of various servers, a number of performance anomalies were encountered that could not be explained based on server architectures or configurations. As a result, it became necessary to probe into the OS in an attempt to understand server behaviour. Without the benefit of working with multiple web servers, running thousands of experiments, and access to the OS source-code, it would have been extremely difficult to identify the source of these anomalies. The remainder of this section presents our experiences trying to understand the performance of web servers executing on Linux and FreeBSD operating systems.

2.1 Experiences with Linux

While contrasting web-server architectures [8, 10], a number of anomalies were tracked into the Linux kernel, where three performance problems were found and fixed. These problems are subtle as they do not cause crashes or result in crippling performance behaviour. Instead, the problems were identified over a period of time due to inconsistency in results and unexplainable performance differences across servers.

2.1.1 Small File Evictions

Kernel versions affected: 2.6.11 to 2.6.21.7
Duration: 02-Mar-2005 to 04-Aug-2007

There was a bug where small files ($\leq$ page size) were being evicted from the file-system cache regardless of their frequency of access. The bug occurred when a change was made to the file-system cache-code to prevent a single, sequential non-page-aligned read of a large file from invalidating a large portion of the file-system cache. However, the mechanism used to detect this behaviour was too coarse; multiple consecutive accesses to the same page in the file-system cache did not update the access flags for that page. Only when a different page in the file is accessed are the access flags updated. This logic results in small files never being marked as accessed after their first access. Hence, these pages are always evicted from the cache regardless of how often or recently the file is accessed.

Situations where the file-system cache fills and must begin evicting pages to disk are potentially affected by this problem. The problem manifests itself through poor disk performance because of less efficient disk access resulting from small, frequently accessed files constantly being reread from disk as opposed to being kept in and served from the cache. The problem becomes acute for applications that place a heavy load on the file-system cache, e.g., web servers, particularly when small files constitute a significant portion of the workload.

The small-file-evictions problem was discovered after publishing performance results [17] using a kernel that contains this bug. The bug was found while conducting subsequent research using a different workload with increased disk I/O. After finding the bug, we reexamined the experiments from the paper and fortunately determined it had only a minor effect on the results, but an effect nonetheless. Hence, we were lucky and the conclusions in the paper are still valid.

2.1.2 Prefetching Disabled

Kernel versions affected: 2.6.12 to 2.6.22.19
Duration: 17-Jun-2005 to 26-Feb-2008

There was a bug where the page-cache read-ahead is disabled for sequential disk-reads when using sendfile with nonblocking sockets, as a result of the kernel misinterpreting the access pattern when reading large files. sendfile is unusual because a call can involve both disk and network I/O. Multiple sendfile calls may be necessary to transmit file data over the network because the size of non-blocking sends are limited by the socket-buffer size. Similarly, the operating system reads a file into the file-system cache from disk in pieces. For large files requiring disk-I/O (i.e., not already in the file-system cache), the socket-buffer size is normally smaller than the amount of file-data read by a single disk-request, so the number of disk accesses required is fewer than the number of network transmissions for the send.

As a result, when transmitting a large file using nonblocking sendfile, the file-access pattern appears random because consecutive sendfile calls do not appear to continue from the end of the last disk I/O but rather continue from some location *within* the last disk read. At this point, the kernel disables page-caching read-ahead for the file and the size of future disk requests for that file become smaller on average. In contrast, for blocking sendfile, only a single sendfile call is required, and since the kernel performs the appropriate tracking, it recognizes file access is sequential, resulting in correct page-cache read-ahead behaviour. We believe this bug resulted from using or adapting a pre-existing kernel function for use with sendfile that originally simply read file data from disk. Unfortunately, assumptions about what constituted sequential access were not correspondingly adapted to recognize the unusual disk access-patterns resulting from sendfile with non-blocking sockets, causing read-ahead to be disabled. This bug was found while trying to understand and explain the differences in performance obtained when using blocking and non-blocking sendfile.

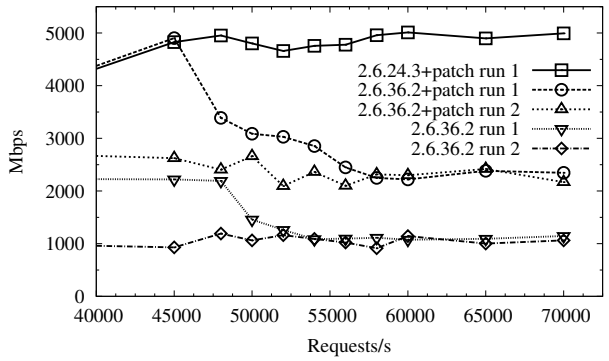

Figure 1: Throughput patched/unpatched kernels

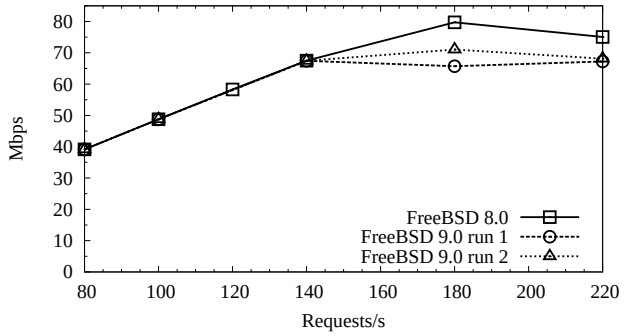

Figure 2: Throughput FreeBSD 8.0/9.0 kernels

2.1.3 Erratic Page Evictions

Kernel versions affected: 2.6.23 until at least 2.6.36.2
Duration: 09-Oct-2007 to at least 09-Dec-2010

There was a bug that results in none of the pages associated with a transmitted file being marked as accessed by the kernel, so the kernel cannot distinguish between recently or frequently accessed pages and other pages in the file-system cache. Therefore, under memory pressure, the kernel may incorrectly evict pages from the file-system cache. When these pages are in the middle of files or frequently accessed, it hampers long contiguous disk-reads and read-ahead buffering, which results in smaller and more random disk requests. This behaviour is manifested as erratic server performance and low disk-throughput.

To correctly mark page accesses for sendfile, we developed a patch for the 2.6.24.3 Linux kernel.[1] This patch provided consistent and repeatable performance measurements, by increasing file-system cache hit rates and improving throughput when reading files from disk. Disk throughput is increased in some of our experiments from approximately 11,000 blocks-in per second (1 block = 1024 bytes) for non-blocking sendfile and 20,000 blocks-in per second for blocking sendfile to approximately 28,000–30,000 blocks-in per second for *both* non-blocking and blocking sendfile. Figure 1 shows some representative experiments to illustrate this I/O problem in later kernels. The patched 2.6.24.3 kernel (line 2.6.24.3+patch run 1) has stable performance and the highest throughput. The unpatched 2.6.36.2 kernel (line 2.6.36.2 run 1) has significantly lower throughput and throughput drops substantially for higher request rates. Repeating the experiment a second time (line 2.6.36.2 run 2) shows a large difference in performance at 45,000 and 48,000 requests per second. After applying our patch for the 2.6.24.3 kernel to the 2.6.36.2 kernel (line 2.6.36.2+patch run 1), throughput is significantly higher compared to the unpatched counterparts. However, throughput is still significantly lower than the patched 2.6.24.3 kernel at higher request rates. Even with the patch, a second run (line 2.6.36.2+patch run 2) has different throughput, showing large variations similar to the unpatched kernel. Without the patch, the 2.6.24.3 kernel exhibited variability problems and poor performance (not shown in the graph), similar to the newer kernel. These experiments indicate there may be an additional performance regression with the newer kernel or that the code has changed enough to make the patch less effective. The variance in throughput for identical experiments, combined with low throughput, directed us to investigate this anomaly further and lead to the bug.

As a result of these performance problems, we used the patched 2.6.24.3 kernel for some of our research [8, 10] because of experience and expertise with this kernel, its stability after patching, and the problems and uncertainty with the newer kernels For other projects we moved to FreeBSD [19, 20].

[1]Our patch for small-file evictions (1st bug) is still in place but the code path for sendfile changed significantly from the earlier to the later kernels (due to splice). As a result, our prior knowledge about sendfile could not be used with respect to the new problem.

2.2 Experiences with FreeBSD

As one might expect, the types of problems we have described are not limited to Linux kernels. When one of our projects started, FreeBSD 8.0 was the most recent stable version of FreeBSD available (released November, 2009). We spent a considerable amount of time using this version and developing expertise with significant portions of the source code. It performed well and offered stable performance across our benchmarks. Because of the length of the project, we were eventually faced with the issue of using a relatively old kernel. We were concerned that it may be viewed as obsolete. In addition, we were also curious to see if a newer kernel might provide better performance and possibly address some issues we were working on. We obtained FreeBSD 9.0 (released January, 2012) and started by conducting a simple experiment to determine if performance was stable across multiple runs of the same experiments and to compare the performance of the older 8.0 kernel with the updated 9.0 kernel. Figure 2 shows that web-server throughput is significantly lower using the FreeBSD 9.0 kernel than the 8.0 kernel. Furthermore, while differences between runs of the same experiment were fairly small on FreeBSD 9.0, there was signifantly more variability across runs than when using FreeBSD 8.0, which was very stable.

3. ENTERPRISE KERNELS

To compare the performance of an Enterprise kernel, the experiments conducted in Section 2.1.3 were run on the

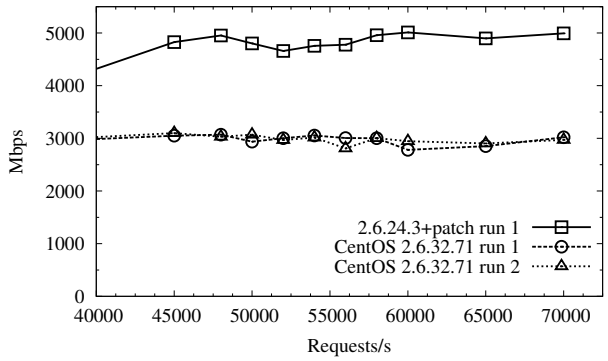

Figure 3: Throughput patched/CentOS kernels

CentOS 6.0 enterprise distribution [4] using kernel version 2.6.32.71. CentOS is an open-source redistributor of a prominent North American Enterprise Linux vendor (currently Red Hat Enterprise Linux). It conforms fully with the upstream vendors redistribution policy and mainly changes packages to remove upstream vendor branding and artwork. CentOS was chosen because it is one of the top two Linux distributions used on Linux web-sites where it has 27% of the market share [25]. Figure 3 shows experimental results are consistent across multiple runs obtaining about 3,100 Mbps; hence, the non-repeatable performance problem did not occur with this distribution. However, its performance is significantly lower than the 5,000 Mbps for the 2.6.24.3 patched kernel and the peak throughput of the patched 2.6.36.2 kernel. While consistent performance is often a first priority for enterprise users, many of these users may be equally concerned with maximum performance. There is no reason why an Enterprise kernel should not provide both.

4. EXPERIENCES FINDING BUGS

All the problems found in Section 2 presented themselves initially as performance differences among different web-server architectures or configurations for a single web server. Debugging performance problems is difficult, especially tracking a performance problem into the Linux kernel. Often the most difficult part is recognizing a performance problem actually exists. In isolation, it is difficult to determine if an application is running reasonably or has a performance issue. We had the benefit of comparing the throughput of several web servers across various configurations and workloads, allowing for the identification of performance anomalies.

Once there is suspicion of a performance problem, the next step is to determine whether the application is defective or if an external factor is causing the problem. In addition to verifying the correctness of the application code, cross-checking application performance with other kernel versions may help in isolating the source of the problem. Finding the source of a performance problem can be challenging as problems often occur only when the application is under full load. This requirement presents two problems. First, it is often necessary to deal with the full complexity of the whole application, which makes isolating the problem more difficult. Second, any debugging and profiling tools being employed may significantly perturb the execution and potentially mask performance issues.

Two common tools for tracking bugs in the kernel are OProfile [16] and SystemTap [22]. OProfile generates dynamic call-graphs along with the execution time spent in each function. We found OProfile was not very helpful because it tended to be too coarse grained. Rather, tools such as vmstat and mpstat were more helpful for our particular web-server work. Unexpected differences in statistics generated by these commands helped to confirm a problem and even suggest the type of problem, e.g., differences in the average blocks-in from disk or the amount time spent waiting for I/O. SystemTap, which offers capabilities similar to DTrace [3], was used to track down the read-ahead problem with non-blocking sendfile in Section 2.1.2, and was helpful with the other problems. As is the case with DTrace, SystemTap is a scripting language useful for instrumenting a running kernel by executing a handler on specified events, such as on entry to or exit from specified kernel functions, allowing the printing of local context-data from within the kernel. With SystemTap, it is possible to dynamically introduce events without having to recompile the kernel or reboot the system, which greatly simplifies the debugging cycle and significantly speeds up the debugging process. Initially, SystemTap was used to understand and track the behaviour of sendfile into the kernel. After identifying the important functions and data structures involved, a subset of the function parameters was printed on entry to these functions. Looking specifically at the output for the functions involved in managing page-cache read-ahead revealed that read-ahead was being disabled with non-blocking sendfile. Based on this information, an examination of the relevant source code revealed the source of the anomaly: a mismatch between the amount of disk I/O and network I/O on calls to sendfile. Without a tool like SystemTap to trace the sendfile call and narrow the search space, finding these problems would have taken significantly longer because the Linux kernel is large and complicated.

5. EXPERIENCES OF OTHERS

Some web sites contain data that tracks the performance of different benchmarks over time (in some cases by kernel version) [11, 13]. Browsing through the collections of benchmarks available on these sites, examples of long and short term performance regressions and improvements can be found. Specifically, the web site "Linux Kernel Performance!" [13] has tracked the performance of several benchmarks executing on Linux kernels from version 2.6.22 to 3.5.0 (at the time of writing). An example of a short-term performance-regression occurs for the Online Transaction Processing benchmark (OLTP) on a 4P quad-core Xeon. Performance drops by approximately 41% from kernel version 2.6.22 to 2.6.23 but improves in subsequent releases until it is back to the 2.6.22 level at 2.6.27, and then improves again in versions up to 2.6.29, but drops slightly and remains consistent until 3.5.0. An example of a longer-term performance-reduction occurs for the benchmark fileio-cfq on a 4P quad-core Xeon. Performance drops by about 40% from kernel version 2.6.31 to 2.6.38 before recovering to slightly above the 2.6.31 level and then remaining relatively consistent until 3.5.0.

Interestingly, changing to a 2P Quad-core Core 2 Duo for the same two benchmarks, OLTP and fileio-cfq, generates differ-

ent performance regressions. For OLTP, the drop is about 28% (41% on the 4P system) starting at 2.6.23, and performance never recovers back to the 2.6.22 level, but instead stays about 7% below 2.6.22 up until 3.5.0. For fileio-cfq, the drop in performance is also 40% but it slowly decreases across a larger range of kernels from 2.6.30 to 3.5.0. If a regression test is performed for fileio-cfq on the 4P system, the 8% improvement may be deemed acceptable, but on the 2P system, the 40% reduction may be deemed unacceptable at 3.5.0. Furthermore, if the range of kernels is reduced to 2.6.32 and 2.6.38 for fileio-cfq, there is little change across both the 2P and 4P, but you would miss the unresolved 20% and 40% performance reductions, respectively. These reductions occurred prior to 2.6.32 but continue to be a problem for this range of kernels. Therefore, it is necessary to track performance across a number kernel versions on different systems to fully understand performance changes.

Some of the benchmarks exhibit huge swings in performance. For example, on the 4P quad-core Xeon system the benchmarks hackbenchpth100 and hackbenchpth150 drop by about 80% from kernel version 2.6.36 to 3.1.0, and than improved by 95% from 3.2.0 to 3.5.0.

Performance regressions can cause problems not only for researchers but also for companies who want to use these kernels in production environments. Conducting benchmarks and reporting the results is crucial for researchers, companies, and other users to allow selecting the best kernel for their needs.

6. CONSEQUENCES

The performance issues raised in the previous sections imply a number of consequences for researchers.

6.1 Reproducing Results

Based on our experience and other published benchmarking results, large performance variations can occur depending on the application and the kernel version. As well, research experiments conducted can be significantly affected based on the particular kernel version that is used. The scientific approach to finding incorrect results or gaining confidence in existing results is for other researchers to reproduce results. Unfortunately, if the original results are verified, it is currently difficult or impossible to publish this work, making the endeavour risky. As in other scientific fields, Computer Science needs to value and publish papers verifying previous results. As well, researchers must provide sufficient methodology, analysis, and data to allow others to perform the same or similar experiments, and hence, reproduce the results. Other scientists have made similar recommendation, both in Computer Science and other fields [1, 2, 6, 7, 12, 14, 18].

6.2 Underlying Cause

The systems research community needs a higher standard for experimentation. Simply reporting performance results (either positive or negative) is insufficient. Based on our experience, it is crucial to find the underlying cause for changes in performance. Experimental results require careful analysis to understand why a change has or has not produced a performance effect, and anomalies in performance results cannot be ignored as they may be "shouting out" that there is an underlying problem. Determining and explaining the root cause for performance results are likely to lead to either an understanding of the observed performance or the discovery of a problem (in some cases, possibly with the kernel).

6.3 Fixing Problems

If unexplainable behaviour suggests a bug, it may be necessary to look into the Linux kernel. Our experience is that finding and fixing a kernel bug is extremely difficult and time consuming, especially because the Linux code-base is large and a quickly moving target. For example, there are many levels of indirection (routine pointers) used in the kernel, so determining what is called and when is difficult. Also, the tool-set for monitoring dynamic execution is low-level and complex to use.

Assuming you find and fix a problem, the next logical step is to have the fix applied to the mainline kernel for the benefit of all. Because the kernel evolves rapidly, it is necessary to obtain the most recent kernel and check if the bug is already fixed. If the bug is still present, it may be necessary to port the fix (again) to the new code base. When the code base has changed significantly, it may be the case that people who fixed the bug in the original version no longer possess the expertise or time required to construct a new fix for the new code. Finally, to create a bug report it is important to write small, stand-alone programs that reproduce the problem, and to submit these programs along with the suggested bug fix. Our experience is that bug reports sent to the kernel-developer mailing-list are not always well received and getting our fixes into the mainstream kernel sometimes required a thick-skin and persistence.

6.4 Kernel Upgrading Problems

Once your research team has established that a kernel works as expected, and generates good, explainable, consistent results, there is the dilemma of moving to the latest version of the kernel because there is a general belief the latest kernel is always better. For researchers, this prejudice appears in the form of reviewers stating that results are not meaningful because the latest kernel is not used. However, based on our experience, bugs we found were not fixed in the new kernel, and new kernels can introduce performance regressions and new problems. Furthermore, new kernels require rerunning and re-validating experiments to re-establish results and gain expertise with the new kernel. This work may take weeks or months, and in the meantime another kernel is released. An important aspect of our work has become explaining and justifying why we are using an older kernel.

We expect that other researchers may have similar experiences. Clearly, progress in the Linux kernel is essential, and the people involved are working actively to do the right things. Additionally, there are cases where switching to the newest kernel is absolutely necessary. However, we do want reviewers, kernel developers, system administrators and users to understand that the latest kernel is not always the best kernel. It is incumbent on all parties to clearly state why an old kernel is better than a new kernel or vice versa. The reason needs to be particular and specific, and not just that the new kernel has fixed a number of bugs and improved performance.

7. RECOMMENDATIONS

Linux kernel developers must employ a systematic, sustained regiment of performance regression testing (to our knowledge this is not currently being done). We understand the difficulties in such an undertaking but expect many of the problems we point out could have been avoided had rigorous performance regression testing been an integrated part of the kernel-development process. Enterprise versions of Linux may perform additional testing and contain customer-driven fixes. However, in the distribution tested (CentOS 6.0) significant performance problems still seem to exist.

Some questions researchers need to ask are:

1. When starting a new project, what version of the kernel should be used and why?
2. When working on a project over an extended period, should the kernel be upgraded and why?
3. If upgrading to a new kernel during a project, does the upgrade change the results significantly, and if so why?
4. How can performance changes be explained by the research that has been done?
5. Have the Operating System changes just worked around or masked problems that were introduced in the newer version and not present in the older version?
6. If the performance of an improved application is slower (or only slightly faster) on a new kernel than the original application was on an older kernel, what is the value of the improvement? Moreover, the improvements made to the application may be simply masking or working around problems in the newer kernel.

Here are some practical suggestions to help answer these questions:

1. Before selecting a kernel, check web sites publishing benchmarks on different kernels and select a kernel with good benchmarks in your research area and avoid those with obvious defects. For example, for reasons explained in Section 5, it is unwise to use version 2.6.23 for workloads that resemble OLTP.
2. After upgrading kernels, run some sanity checks for comparison. If performance improves or degrades, try to determine why. This requires expertise, determination, and time, with no guarantee of success.
3. In general, experiments must be run multiple times to check for variability. If there is variability, explain why and report confidence intervals.
4. When conducting experiments, appeal to your intuition. Researchers sometimes become blind when it comes to obtaining results. If results are significantly better or worse than expected, figure out and explain why.
5. Ensure the experimental environment is sound. For example: address-space randomization (a computer security technique) may cause variations in results; Security-Enhanced Linux (SELinux) may reduce performance for some workloads because of its security checking; processor dynamic-frequency-scaling may cause variations in results due to changes in clock frequency. Therefore, it may be appropriate to enable or disable some of these mechanisms depending on the particular experiment.

8. SUMMARY

Linux is an excellent platform for both conducting research and as a production environments. Furthermore, the Linux kernel developers are doing an excellent job building an innovative and robust operating-system. This paper (and many others) would not have been written without their dedication and effort. Working in conjunction with the kernel developers are university and industrial researchers who use Linux to demonstrate new ideas, approaches, and techniques across a spectrum of disciplines. A goal of these researchers is to see their technologies move from the laboratory into production. One reason for Linux's success is its continuous inclusion of innovations and improvements resulting in frequent release cycles.

This paper highlights the issues and the problems that result as the kernel evolves. It is unavoidable that changes must be made to fix bugs, add new features, enhance maintainability, improve scalability, or increase performance. In many cases, kernel developers must make complex decisions regarding tradeoffs among these changes, which can affect different benchmarks and applications in different ways on different systems. Coupled with the relentless pace of change, bugs and performance regressions can occur in newer versions of the kernel.

Our key points are:

1. For experimentation and production systems, the latest version of Linux is not necessarily the best version to be using, and researchers, reviewers, kernel developers, and users need to think through and understand the pros and cons of different kernel versions. However, after due consideration, it may be that the most recent version of the kernel is the best version to be using.
2. In light of the significant performance bugs we found and the time periods over which they have been present, it is possible that performance results published across an extended time period should be reevaluated.
3. More papers need to provide a deep analysis of their experimental results. While such analysis is time consuming and difficult, it provides understanding of where the benefits come from and insights into applicability beyond the scope of the paper.
4. The computer-systems research-community needs to embrace the scientific approach of publishing papers that reexamine previous work (in non-trivial ways) to either confirm or refute their results. This effort should include different hardware configurations, different operating systems, and different workloads.

9. ACKNOWLEDGMENTS

We thank Jim Summers for his work in obtaining and graphing the FreeBSD 8.0 versus 9.0 results and the anonymous reviewers for their suggestions for improving this paper. Funding for this project was provided by the Natural Sciences and Engineering Research Council of Canada.

10. REFERENCES

[1] M. Baker. Independent labs to verify high-profile papers. *Nature | News*, August 2012.

[2] S. M. Blackburn, A. Diwan, M. Hauswirth, P. F. Sweeney, J. N. Amaral, V. Babka, W. Binder, T. Brecht, L. Bulej, L. Eeckhout, S. Fischmeister, D. Frampton, R. Garner, A. Georges, L. J. Hendren, M. Hind, A. L. Hosking, R. Jones, T. Kalibera, P. Moret, N. Nystrom, V. Pankratius, and P. Tuma. Evaluate collaboratory technical report #1: Can you trust your experimental results?, February 2012. http://evaluate.inf.usi.ch/technical-reports/1.

[3] B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal. Dynamic instrumentation of production systems. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '04, Berkeley, CA, U.S.A., 2004. USENIX Association.

[4] CentOS: The community enterprise operating system. http://www.centos.org.

[5] J. Corbet, G. Kroah-Hartman, and A. McPherson. Linux kernel development: How fast it is going, who is doing it, what they are doing, and who is sponsoring it, Mar. 2012. http://go.linuxfoundation.org/who-writes-linux-2012.

[6] Evaluate Collaboratory. Experimental evaluation of software and systems in computer science. http://evaluate.inf.usi.ch/.

[7] D. G. Feitelson. Experimental computer science: The need for a cultural change, December 2006. http://www.cs.huji.ac.il/ feit/papers/exp05.pdf.

[8] A. S. Harji. *Performance Comparison of Uniprocessor and Multiprocessor Web Server Architectures*. PhD thesis, University of Waterloo, 2010. http://hdl.handle.net/10012/5040.

[9] A. S. Harji, P. A. Buhr, and T. Brecht. Our troubles with Linux and why you should care. In *Proceedings of the Second Asia-Pacific Workshop on Systems*, APSys '11, pages 2:1–2:5, New York, NY, USA, July 2011. ACM.

[10] A. S. Harji, P. A. Buhr, and T. Brecht. Comparing high-performance multi-core web-server architectures. In *Proceedings of the 5th Annual International Systems and Storage Conference (SYSTOR 2012)*, pages 2:1–2:12, New York, NY, USA, June 2012. ACM.

[11] M. Larabel. *Five Years Of Linux Kernel Benchmarks: 2.6.12 Through 2.6.37*. Phoronix Media, Nov. 2010. http://www.phoronix.com/scan.php?page=article-&item=linux_2612_2637&num=1.

[12] J. Lehrer. The truth wears off: Is there something wrong with the scientific method? *The New Yorker*, December 13, 2010.

[13] Linux kernel performance! http://kernel-perf.sourceforge.net.

[14] J. N. Matthews. The case for repeated research in operating systems. *SIGOPS Operating Systems Review*, 38:5–7, April 2004.

[15] Netcraft, Jan. 2012. https://ssl.netcraft.com/ssl-sample-report/CMatch/oscnt_all.

[16] OProfile, 2012. http://oprofile.sourceforge.net.

[17] D. Pariag, T. Brecht, A. Harji, P. Buhr, and A. Shukla. Comparing the performance of web server architectures. In *Proc. of the 2nd ACM SIGOPS/EuroSys Conf. on Computer Systems*, pages 231–243. ACM, Mar. 2007.

[18] C. Small, N. Ghosh, H. Saleeb, M. Seltzer, and K. Smith. Does systems research measure up? Technical report, Harvard University, TR-16-97, 1997.

[19] J. Summers, T. Brecht, D. L. Eager, and B. Wong. Methodologies for generating HTTP streaming video workloads to evaluate web server performance. In *5th Annual International Systems and Storage Conference (SYSTOR)*, pages 13:1–13:12, 2012.

[20] J. Summers, T. Brecht, D. L. Eager, and B. Wong. To chunk or not to chunk: Implications for HTTP streaming video server performance. In *22nd SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 15–20, Toronto, Canada, June 2012.

[21] SUSE linux enterprise server for system z: The market leader for linux on ibm mainframes, May 2010. http://www.novell.com/docrep/2010/05/-SLES_for_system_z_market_share_leader.pdf.

[22] The SystemTap home page, 2010. http://sourceware.org/systemtap/.

[23] W. F. Tichy. Should computer scientists experiment more? *Computer*, 31(5):32–40, 1998.

[24] Top 500 supercomputer sites. http://www.top500.org/statistics/list, Category→Operating System Family→Submit.

[25] Web Technology Surveys. Historical trends in the usage of Linux versions for websites, Aug. 2011. http://w3techs.com/technologies/history_details/os-linux.