Our Troubles with Linux 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 provides researchers with a full-fledged operating system that is widely used and open source. However, due to its complexity and rapid development, care should be exercised when using Linux for performance experiments, especially in systems research. 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 In addition, the rapid kernel development cycle means that experimental results can be viewed as out of date, or meaningless, very quickly.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 the Linux kernel as a platform 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 has existed 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.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Distributed Systems—*client/server*; D.4 [**Operating Systems**]: Reliability—*verification*; D.4 [**Operating Systems**]: Performance—*measurements*

General Terms

Experimentation, Measurement, Performance

Keywords

Linux, bugs, web servers, regression testing

2nd ACM SIGOPS Asia-Pacific Workshop on Systems (APSys 2011) July, 2011, Shanghai, China

1. INTRODUCTION

Linux is a boon to academic systems-researchers because it provides an open-source platform to make improvements to and evaluate the performance of systems that are used in production. Prior to Linux, OS, networking and database researchers were often at the mercy of OS vendors with respect to developing and evaluating new OS 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 Linux has forced some OS 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.

However, due to its complexity and rapid development, using Linux for systems research also has its drawbacks. 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.

The main contribution of this paper is demonstrating that significant performance problems exist in multiple Linux kernels. As well, we argue that:

- some performance results published over an extended time-period need to be re-examined due these performance issues.
- to encourage good science, publishing papers that validate prior research results is essential.
- experimental results need careful analysis to understand and explain why a change has or has not produced a performance effect.

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

Finally, we make a number of recommendations for performing sound experimental performance evaluations.

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 architecture affect performance. During detailed comparisons of various servers, a number of performance anomalies were encountered that could not be explained based on server architectures or configurations [1]. The anomalies were eventually tracked into the Linux kernel, where three Linux performance problems were found. These problems are subtle as they do not cause crashes or typically result in crippling performance behaviour. Without the benefit of working with multiple servers and access to the Linux source-code, it would have been difficult to identify these anomalies.

2.1 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 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 sitting in 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 [4] 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.

2.2 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 non-blocking 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, with the size of each piece determined by the disk-I/O scheduling algorithm. 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, for nonblocking sendfile, the file-access pattern appears random because consecutive sendfile calls, when transmitting a large file, 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.

2.3 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 may exist a sendfile bug in the most recent Linux kernel (present the last time we checked in December 2010). This bug 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 improv-

[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.

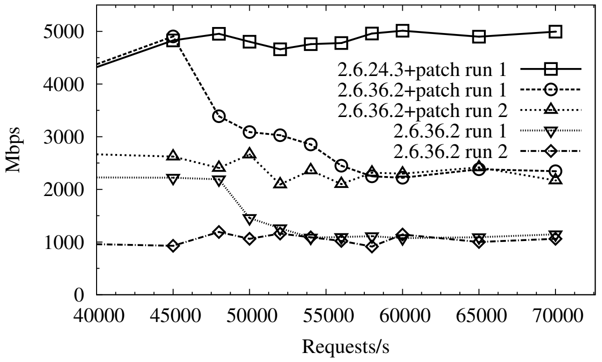

Figure 1: Throughput for patched and unpatched kernels

ing 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 recent 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 similar problems (not shown in the graph). 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, for our research work we either: use the patched 2.6.24.3 kernel because of our experience with this kernel, its stability after patching, and the problems and uncertainty with the newer kernels; or are trying FreeBSD on one new project.

3. EXPERIENCES FINDING BUGS

Debugging performance problems is difficult, especially tracking a performance problem into the Linux kernel. Sometimes, the most difficult step is to recognize that a performance problem actually exists. In isolation, it is difficult to determine if an application is running reasonably or if there is a problem with its performance. In our work, we had the benefit of comparing the throughput of several web-servers across various configurations and workloads allowing us to identify performance anomalies. Finding the source of a performance problem can be challenging as problems often occur only when the application is under full load, when debugging and profiling tools may significantly perturb the environment.

Two common tools for tracking bugs in the kernel are OProfile and SystemTap. 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, we found tools such as vmstat and mpstat to be more helpful for our particular web-server work. Unexpected differences in their statistics helped to confirm a problem and even suggested the type of problem. SystemTap was used to track down the read-ahead problem with non-blocking sendfile, and helpful with the other problems. It is a scripting language useful for instrumenting a running Linux kernel by executing a handler on specified events, such as on entry to or exit from specified kernel functions, allowing the printing of local data. 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.

4. EXPERIENCES OF OTHERS

Some web sites contain data that tracks the performance of different benchmarks over time (in some cases by kernel version) [3, 2]. 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!” [3] has tracked the performance of several benchmarks executing on Linux kernels from version 2.6.22 to 2.6.38 (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 45% from kernel version 2.6.22 to 2.6.23 and improves in subsequent releases until it is back to the 2.6.22 level in version 2.6.25. An example of a longer-term performance-reduction occurs for the benchmark fileio-cfq on a 4P quad-core Xeon. Performance drops by about 30% from kernel version 2.6.31 to 2.6.32 and performance of this benchmark has not improved from that level with subsequent releases of the kernel (up to 2.6.38).

Interestingly, changing to a 2P Quad-core Core 2 Duo for the same two benchmarks on the two kernel releases (OLTP on 2.6.23 and fileio-cfq on 2.6.32) generates different performance regressions. The degradations are about 20% for OLTP (45% on the 4P system) and only 5% for fileio-cfq (30% on the 4P system). If a regression test is performed on the 2P system, the 5% reduction may be deemed acceptable, but if performed on the 4P system, the 30% reduction may be deemed unacceptable. Furthermore, if the range of kernels is altered to 2.6.29 and 2.6.31, there is a 20% reduction on the 2P, but a 3% increase on the 4p. 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 benchmark hackbenchpth150 improved by about 2,000% from kernel version 2.6.25 to 2.6.26. Unfortunately, those gains disappeared with the release of 2.6.36 and have stayed at the reduced level (to version 2.6.38).

Performance regressions cause problems not only for researchers but also for companies. Companies want to use these kernels in production environments to conduct and report results of important benchmarks using a version of the kernel without performance problems.

5. CONSEQUENCES

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

5.1 Problems in Published Papers

A number of papers across many disciplines over multiple years may contain incorrect performance results. Based on our experience, papers involving significant I/O may be affected. As well, based on the benchmarking results across different kernel versions, performance variations occurred across different parts of the kernel, so the scope of affected papers/results could be larger than what we report on. The scientific approach to finding incorrect 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 that verify previous results.

5.2 Underlying Cause

Based on our experience, it is crucial to find the underlying cause for performance results. 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 because 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). Simply reporting performance results (either positive or negative) is insufficient.

5.3 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 no longer possess the expertise or time required to construct a new fix. Finally, to create a bug report it is important to write small, stand-alone programs that reproduced 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.

5.4 Kernel Upgrading Problems

Once your research team has established a working kernel, which 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, which 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.

6. POSSIBLE 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.

Some questions researchers need to ask are: When starting a new project, what version of the kernel should be used and why? When working on a project over an extended period, should the kernel be upgraded and why? If upgrading to a new kernel during a project, does the upgrade change the results significantly, and if so why? How can performance changes be explained by the research that has been done?

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 4, 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 why.

5. Ensure the experimental environment is sound. For example: address-space randomization (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.

7. 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 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) 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. It may also be the case 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 very likely that performance results published across an extended time period need to 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.

8. ACKNOWLEDGMENTS

Funding for this work was provided by the Natural Sciences and Engineering Research Council of Canada. We would like to thank the anonymous reviewers for helpful comments and suggestions on an earlier version of this paper.

9. REFERENCES

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

[2] 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.

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

[4] 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.