Proceedings of the Linux Symposium

Volume Two

July 19th–22nd, 2006
Ottawa, Ontario
Canada

Automatic System for Linux Kernel Performance Testing

Alexander Ufimtsev
University College Dublin
alexu@ucd.ie

Liam Murphy
University College Dublin
Liam.Murphy@ucd.ie

Abstract

We introduce an automatic and open kernel testing system. We argue that only by opening a test system to the community and aggregating the results from a variety of sources can one get a comprehensive picture of the kernel's performance status. Our system can also help identifying problems with specific parts of code whether it is a device driver, some other module, or platform-specific code. Design of both client and server parts of the system is described. Since the system is open, specific emphasis in the client part is placed on successful automation and configuration of the testing process. The emphasis of the server part is placed on regression detection and accidental/malicious input elimination. Current implementation status is presented.

1 Introduction

Testing is an integral part of any quality software development process. Some software development practices even dictate the necessity of writing tests prior to writing an actual method, function, or process for it. However, some of the extra functional requirements can only be checked during integration or even system testing. *Performance* is one of the extra functional requirements that is difficult to check outside of a proper testing environment. Automating system performance tests require a lot of provision and foresight from the authors, taking into account all unusual and unpredicted situations that might happen during the tests. Watchdogs and exception handling are required for specific benchmarks, buggy code, crashes, file corruption—a lot of things might go wrong when working with unstable code. The matter becomes even more difficult when trying to do performance testing of the kernel. Since the kernel is not a process that can simply be killed and restarted, but rather a host to the processes itself, the ability to handle test exceptions gracefully is quite limited. Of course, it is possible to use virtualization methods, such as User-mode Linux [1], VMware [3], or Xen [2] to improve control over the whole test process. However, the performance results obtained using virtualization are not authentic for the actual hardware, but rather for the specific virtualization kernel is tested with. Comprehensive automatic kernel tests help to prevent instability issues and performance regressions, while lack thereof is considered to be a significant contributor to the kernel quality problem.

2 Related Work

Two of the most well-known projects dealing with Linux kernel performance are Automatic test system by Bligh [4] and Linux Kernel Performance project by Chen *et al.* [5]. The former is a widely publicized automated system that performs a variety of tests on a number of high-end machines with a smaller set of test tools. The latter is a less known (semi-) automated system that utilizes fewer hardware resources but provides a more comprehensive set of benchmarks. Other related work includes *kerncomp* [6] and Open Source Development Lab's Linux Stabilization and Linux Testing [7] and Linux 2.6 Compile Statistics [8]. Most of the kernel testing statistics are from available "big iron" machines. Though undoubtedly useful, the test results produced by these projects are quite unrepresentative, since they tend to test kernel performance on very specific hardware with very specific configurations. As authors note themselves, "[p]erformance tests and ratings are measured using specific computer systems and/or hardware and software components and reflect the approximate performance of these components as measured by those tests." [5]

3 Proposed Solution

The Linux kernel can run on all kinds of platforms supporting a variety of hardware, and has a huge number of configurable parameters. We argue that community involvement is necessary to get a comprehensive picture of kernel performance the same way kernel is being developed itself. By analyzing test data performed on various types of hardware with statistical and data-mining methods, it is possible to construct a detailed picture of kernel behavior on different computer architectures, configurations, and devices.

3.1 System Requirements

The goal of our project is an extensible and easy-to-use automatic testing system that downloads, compiles, installs, and runs performance tests of kernel branch snapshots. The resulting data are sent to a server and then made available to developers via a web interface. If a regression is found, our system can pinpoint the problem down to specific architecture, device driver, submitter, and theoretically, the specific configuration that causes it.

The following features were identified as essential for this open and community-driven testing system:

- *Security* – system should be secure. We need to be able to identify and isolate erroneous and malicious input so the overall results are not affected. Therefore required user registration needs to be enforced for use of the system.
- *Simplicity* – system should be as simple as possible to appeal to a wider range of audience. Ease of installation and configuration, ease of result interpretation.
- *Compatibility* – system should be able to run in a similar fashion on a variety of architectures.
- *Extendability* – system should be configurable and extendable to be able to include more tests, test different kernel branches, be able to send data to various servers, if necessary.
- *Stability* – system should try to recover from various errors during testing and be able to avoid them, if necessary.
- *Speed* – developers and testers should not wait too long before they can see the results.

3.2 Architecture Overview

The high-level overview of the system is presented in Figure 1. Multiple client machines that run on various hardware perform tests submit the results to the submission management module of the server. Results are processed and stored in a database backend. Analysis and Presentation module processes data and makes the information available to the developers. The following sections discuss the architecture of client and server.

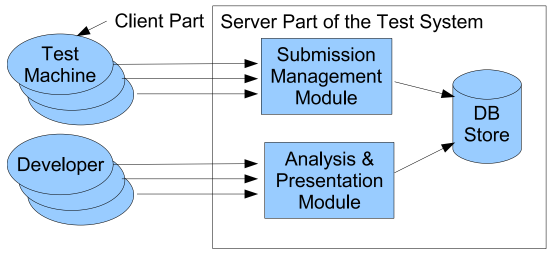
Client Part
Server Part of the Test System
Test Machine
Developer
Submission Management Module
Analysis & Presentation Module
DB Store

Figure 1: High-level system overview

4 Client Architecture

Client architecture is presented in Figure 2. Installation of the kernel image takes place after successful compilation. Current implementation of the system does not support modules and requires a single monolithic kernel that is installed as a single file. After booting, the system gives an administrator a chance to intervene and interrupt the tests within a certain time before the tests actually start. After test runs the results are parsed, signed with a key that tester obtained from the server, and sent to that server. Once the results are sent, the client updates repositories and looks for changes. If no changes are introduced to the tested repositories, the computer waits a certain amount of time and tries to update again. Otherwise the branch selection decision is made based on the updated source, amounts and dates of previous tests, and whether tests failed or succeeded. Upon successful compilation, the kernel is installed and booted into, otherwise it waits and updates the repositories again. As an alternative to pure waiting before attempting to update the repositories again, the system could perform other tests using the same source code and kernel to determine current results/stability. The behavior depends on the number of source trees for testing configured in the test client and the frequency of their updates.

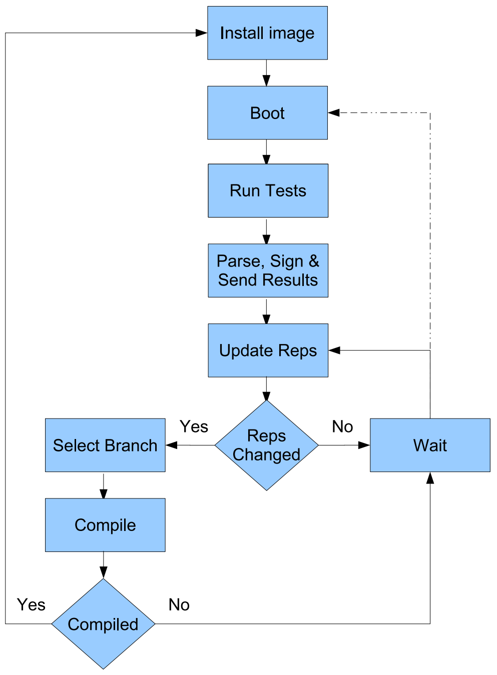

Figure 2: Client Logic

4.1 Tests

To be compatible with the majority of architectures, we plan to use available standard benchmarking tools. Martin [9] provides a good overview of the available tools. The tests were separated into "mini" and "standard" packages. The "mini" package needs to be used on machines with limited resources, where *gcc* is not available. "Standard" also contains tools that require use of *gcc* for proper functionality. Both "mini" and "standard" sets are easily extendable and customizable, if necessary.

Long Tests vs. Short Tests. On one hand, developers would like to know about the test results as soon as possible. On another hand, tests are useless if they do not provide certain accuracy. Long tests are randomly introduced to the system to check the validity of the short tests. Approximately every 10–15 runs, instead of a short run, the system is going to perform a long run, using longer startup, cooldown, and measure time parameters for different tests.

Watchdogs. Watchdogs are essential for stable functionality of the tests. Quite a few tests can misbehave when working on a non-stable kernel. Lockups or race conditions, segmentation faults or file corruptions—watchdogs should try every possible way to recover the tests and continue. Currently we can prevent lockups of the test processes and secure deletion of the temporary files due to abnormal program terminations. It is not possible for now to recover from kernel panic, for instance. See Section 6 for more information.

4.2 Snapshot Selection

Selection of a snapshot for subsequent compilation and testing depends on a number of things. First, a user can select the kernel branches that he or she is interested in testing. By default each branch has an equal priority, which can be changed at configuration time. Other factors that also affect the choice are the number of previously performed compilations, their overall success rate, and date and time of the last compilation attempt.

5 Server Architecture

The centralized server software manages data submissions, data aggregation, and results presentation.

5.1 Data Submission Management

Server checks the signature of the sent dataset to determine whether it should be accepted or rejected. Unsigned submissions are accepted by default, but this policy can be reconfigured by server admins. The following information is sent by clients to a server by default:

- test results
- `.config` file
- snapshot of `/proc`
- mount output
- gcc version (for standard tests)
- glibc version

All this information, except for the test results is only sent to the server if allowed by admins to do so and changed from a previous run due to configuration or software and hardware upgrade. It is used for datamining and problem localization as explained in the Section 5.2. Other information can be added, if necessary.

5.2 Data Analysis and Aggregation

Data analysis is performed for filtering out faulty or erroneous data and for regression detection. Since the amount of available hardware and software options is too great, we are mostly interested in deltas (Δ), not the absolute numbers. Below is an algorithm used for data analysis.

1. *Calculate personal cut-off average (avg) Δ, personal sliding avg Δ.* This is a necessary part of the calculation. We work with deltas since absolute numbers are almost useless due to variety of hardware the tests run on.

2. *Compare them to common cut-off avg Δ and sliding avg Δ. If there is no significant difference, update common. Go to END.* Now we know that there is probably no problem with the current build. It is not interesting, so we finish analysis. Otherwise—keep going.

3. *Compare the results to the rest of the similar parameters mentioned in the Section 5.1.* If there is a number of results submitted already that deviate from the average results and some of the parameters are common, we localize our search and mark these parameters for further inspection. If no similar configurations show the same type of deviation, we mark results as suspicious for further analysis.

4. *We inspect the submissions to the source tree at this update to see any similarities between localized potential problems and patches submitted to that area of the source tree.* If there are any matches, there is a high probability that those changes introduced a detected regression (or improvement, for that matter). Report our findings.

5.3 Results Presentation

Once newly arrived data are analyzed, the presentation module updates the result pages. The summary page contains all the important development for the projects and highlights possible problems. The rest of the statistics are also available.

6 Potential Problems

There are a number of potential problems that currently do not have a solution. First, running a development version of the kernel, compiling and booting into that kernel creates an instability point in the system. There is little that can be done for majority of architectures if kernel panics during the boot or during the tests. Some hardware allows a computer to be rebooted remotely; the others require human intervention. Hardware watchdogs can help to solve this problem, though. And even if a system is restarted automatically, it needs to be able to boot into a safe and stable kernel instead of the buggy one. This would require modification of the bootloader for majority of architectures. And even if the safe kernel is chosen, there is always a possibility that tests running in the unstable kernel have corrupted the filesystem.

Another major concern is an accuracy of the tests. Aggregating a huge amount of data, some of which might be malicious, is a tedious task. Also, the system needs to be very precise in order to be useful. Even 99% accuracy is not enough since the remaining 1% of false positives would create a frustration with users that would at some point just stop using the system.

7 Current Status and Future Work

Currently the system is being implemented with clients up and running on three architectures: x86, alpha, and ppc. Preliminary result aggregation of the results is also implemented on the server side of the test suite. We are looking into ways of solving open questions mentioned in Section 6. One of the possible ways of battling file system corruption is a way suggested by Instalinux [10] through creation of a customized bootable CD while the current system state is stored on the server in members profile.

Since the system is now in the active development stage, we plan to introduce more features into it and make its presentation (as well as open it to the community) during the presentation.

8 Conclusion

The design and current implementation status of an automatic system for Linux kernel testing was presented. Both client and server parts of the system were discussed, processing algorithms mentioned. Potential problems and possible ways of solutions were also addressed.

The advantage of this system over already existing ones is that the system allows developers to have a coherent view on kernel performance. The accepted results come from various hardware architectures and software configurations with the help of the Linux community.

9 Acknowledgment

Authors would like to thank Valentin Chulkov and Niamh Mahon of UCD for their help with the project.

References

[1] User-mode Linux. `http://user-mode-linux.sourceforge.net`. (2006)

[2] XenSource. `http://www.xensource.com`. (2006)

[3] VMware. `http://www.vmware.com` (2006)

[4] M.J. Bligh. Automated Linux Testing. `http://test.kernel.org`. (2006)

[5] Chen, K. and Chen, T. The Linux Kernel Performance Project. `http://kernel-perf.sourceforge.net`.

[6] Wienand, I. and Williams, D. Tools for Automated Regression Testing of the Linux kernel kerncomp.sourceforge.net (2006)

[7] Open Source Development Labs. Linux Stabilization and Linux Testing. `http://osdl.org/projects/26lnxstblztn/results`. (2006)

[8] Open Source Development Labs. Linux 2.6 Compile Statistics. `http://developer.osdl.org/cherry/compile` (2006)

[9] J. Martin. Linux Test Tools. `http://ltp.sourceforge.net/tooltable.php` (2006)

[10] Instalinux. `http://www.instalinux.com`.