Leap Second Testing on MontaVista Carrier Grade Edition 7 (CGE7)

Prem Karat (pkarat@mvista.com)

Abstract

A leap second will be inserted at the end of June 30, 2015 at 23:59:60 UTC. Previously, on 30 June, 2012 a leap second was added. Due to this, Linux servers displayed a notable tendency to misbehave during the leap second event at the end of the day on June 30. The problem often presented itself as abrupt and sustained load spikes on the affected machines. The bug that caused this behaviour last time has been tracked down and fixed by Linux Kernel community. This paper tries to cover different methods to mitigate any unavoidable bugs that may or may not be triggered by the leap second insertion. The objective of the paper is to provide a test methodology/procedure to setup a test environment so that sysadmin/users can test and report any issues, well in advance on what happens when a leap second is inserted. This way they can be well prepared to brace up for any issues due to leap second insertion. This paper does not to provide a definite method of avoiding any bugs due to leap second insertion.

Keywords— Leap second, ntp, ptp, linux kernel, CGE7, carrier grade linux

I. INTRODUCTION [1]

Leap seconds are scheduled by the International Earth Rotation Service (IERS) whenever the difference between true earth rotation and UTC time scale reaches a certain limit. Whenever a leap second has been scheduled by the IERS, a warning must be disseminated to time keeping devices so that clocks become aware of the scheduled leap second early enough to be able to handle the leap second properly. There are different ways to propagate leap second warnings, and different ways to handle leap seconds, and thus a number of pitfalls causing unexpected results and potential malfunctioning.

Leap second warnings can be propagated by different timing signals, protocols, etc. Also, there are different implementations how leap seconds are handled, which especially affect the sequence of timestamps across the leap second event. There is no exact specification how leap seconds are to be handled by clocks providing time with resolution below 1 second. If a leap second is deleted then one second can simply be skipped, i.e. time could just be stepped forward by 1 second. If a leap second is inserted, which is usually the case, a rule is required how time should be incremented during the inserted leap second.

Most articles about leap seconds only refer to the way leap seconds are counted for human readable date and time. If a leap second is deleted then the 59th second of the last minute before UTC midnight is just skipped, and an inserted leap second is simply counted as second 60 before midnight. This behavior is perfectly acceptable for wall clocks and displays with only second resolution:

Table 1: Enumeration Of Seconds Over A Leap Second

20120630 23:59:57
20120630 23:59:58
20120630 23:59:59
20120630 23:59:60 < leap second
20120701 00:00:00
20120701 00:00:01
20120701 00:00:02

As can be seen, if the times in Table 1 are normalized then 60 seconds yield 1 minute, which is carried over to the minutes field. Thus the minute count reaches 60 and rolls over to 0 with a carry to the hours and so on, so the resulting normalized time is exactly the same as the first second of the next day. This is exactly what happens with computer clocks, etc., which simply count the number of seconds since a given epoch. If they run on UTC as proposed by the POSIX standard the second count is the same at the beginning and end of the inserted leap second. So how to distinguish between both, and how to enumerate time stamps taken in short intervals during an insert leap second?

There are different approaches how to do this, and each approach has its own advantages and disadvantages. Usually the exact behavior is implemented in the operating system kernel

a. Step Time Back At The End Of The Leap Second

b. Step Time Back At The Beginning Of The Leap Second

c. Stop Time Counting For 1 Second

d. Increment Time LSB On Request

e. Slewing Time Half Speed Over 2 Seconds

f. Google Leap Smear

Known Bugs in the past:

There are some popular bugs which caused malfunctioning on related systems or even let the affected systems stop immediately, so the services running on those systems became unexpectedly unavailable. Some of these bugs were

1. Linux Kernel Deadlock

While holding a kernel lock, the Linux kernel's leap second handling code tried to log a message about the leap second insertion, which in turn tried to acquire the same kernel lock. On busy kernels this could lead to a deadlock so the machine completely stopped working when the leap second was inserted.

2. Leap Second Caused High CPU Load on Linux Systems

After the leap second on June 30, 2012, CPU load increased to 100 % continuously which caused significantly increased overall power consumption in data centers.

All the approaches and known bugs above are covered in detail in the paper 'Technical Aspects of Leap Second Propogation and evaluation by Martin Burnicki' [1].

In this paper, we will try to focus on 3 aspects.

a. How MontaVista CGE7 kernel handles leap second insertion on various platforms.

b. How the NTP daemon (client) just passes a leap second down to the MV CGE kernel, which handles the leap second.

c. Steps taken by MontaVista engineering team to cover PTP.

The following platforms were covered during our testing

a. armeb-cortex-a15-3.10-1.4

b. cavium-octeon2-3.10-1.4

c. freescale-8548cds-3.10-1.4

d. x86-generic-64-3.10-1.4

II. MV CGE7 KERNEL TESTING [2]

The kernel's core time is kept in a timespec structure:

```
struct timespec {
        __kernel_time_t tv_sec;     /* seconds */
        long tv_nsec;               /* nanoseconds */
};
```

It is, in essence, a count of seconds since the beginning of the epoch. Unfortunately, that count is defined to not include leap seconds. So when a leap second happens, the system time must be explicitly corrected; that is done by setting the system clock back one second at the end of that leap second. The code that handles this change is quite old and works pretty much as advertised. It is the source of this message that most Linux systems should have (in some form) in their logs:

[179712.833890] Clock: inserting leap second 23:59:60 UTC

The kernel's high-resolution timer (hrtimer) code does not use this version of the system time, though — at least, not directly. Instead, hrtimers have a couple of internal time bases that are offset from the system time. These time bases allow the implementation of different clocks; the "realtime" clock should adjust with the time, while the "monotonic" clock must always move forward, for example. Importantly, these timer bases are CPU-specific, since realtime clocks can differ between one CPU and the next in the same system. The hrtimer offsets allow the timer subsystem to quickly turn a system time into a time value appropriate for a specific processor's realtime clock.

If the system time changes, those offsets must be adjusted accordingly. There is a function called clock_was_set() that handles this task. As long as any system time change is followed by a call to clock_was_set(), all will be well. The problem, naturally, is that the kernel failed to call clock_was_set() after the leap second adjustment, which certainly qualifies as a system time change. So the hrtimer subsystem's idea of the current time moved forward while the system time was held back for a second; hrtimers were thereafter operating one second in the future. The result of that offset is that timers started expiring one second sooner than they should have; that is not quite what the timer developers had in mind when they used the term "high resolution."

When leap second occurs, non-monotonic clock sources, for example, CLOCK_REALTIME is set back one second. At interrupt time (T), the hrtimer code expires all CLOCK_REALTIME based timers set for T+1s and before, causing early expirations for timers between T and T+1s since the hrtimer code's sense of time is one second ahead. This causes all TIMER_ABSTIME CLOCK_REALTIME timers to expire one second early. More problematically, all sub-second TIMER_ABSTIME CLOCK_REALTIME timers will return immediately. If any such timer calls are done (as commonly done with futex_wait or other timeouts), this will result in infinite loops and cause load spikes in those applications. The bug in CLOCK_REALTIME hrtimer was fixed (by kernel community) by calling clock_was_set().

Some applications are poorly written to use time differential on non-monotonic clock sources. On a leap

second, these would see time go backwards and calculate a negative differential, resulting in incorrect behavior, from lockups to infinite loops. A typical bug (like the one above) in the non-monotonic clock sources will be out of testing scope.

The work done by clock_was_set() must happen on every CPU, since each CPU has its own set of timer bases. That's not something that can be done in atomic context. So John's patch detects a call in atomic context and defers the work to a workqueue in that case. With this patch in place, the kernel's leap second handling should work again

MontaVista QA engineering team has designed a semi-automated test suite that tests Linux Kernel for all the regression bugs mentioned above. The code for Leap Second insertion/deletion is designed and developed by the Linux Kernel Community [3] and AmadeusITGroup [4], but the code has been modified with improvements to suit the test execution. Also MV QA team has designed a test methodology which includes testing the Kernel Leap Second handling code for robustness and stability under severe load. The leap second kernel testing can be classified into 3 categories

1. Test for basic leap second insertion/deletion
2. Test for leap second kernel deadlock
3. Stress test for leap second insertion/deletion under severe load that includes running the following in parallel
   a. Futex Stress [5]
   b. Realtime application load using cyclictest
   c. Scheduler stress using hackbench
   d. Filesystem stress using bonnie++

This would test for any high CPU load issues on the system/target under test.

Test Results:

None of the platforms showed any regression failures (kernel deadlock or high CPU load issues) noted in the past and all platforms were able to insert/delete the leap second and showed the following expected message in kernel buffer (dmesg)

[XXX.XXX] Clock: inserting leap second 23:59:60 UTC

[XXX.XXX] Clock: deleting leap second 23:59:59 UTC

III. MV Network Time Protocol (NTP) Testing [1]

The Network Time Protocol (NTP) also supports announcement of a negative or positive leap second. However, this is coded in a way that only one type of leap second can be announced at the same time, which avoids potential errors as with PTP. For the reference implementation the exact behavior of leap second handling varies with the software version, e.g. the time interval at which a leap second warning is accepted prior to the event, the conditions under which a leap second warning is accepted if there are several potential sources for a leap second announcement, like one or more refclocks, one or more upstream NTP servers, and optionally the NIST leap second file.

NTP package version in MV CGE7 is v4.2.6p5 which supports the kernel discipline and in this case ntpd just passes a leap second down to the CGE7 kernel which handles the leap second. The following are the different approaches to handle the leap second through NTP.

1. Allow NTP to pass the leap second flag down to kernel which handles the leap second.

2. Disable leap second flag in kernel, disable NTP during leap second and at some point in the future, sync the system time with rest of the world (using tools like tickadj and adjtimex)

3. Disable leap second flag in kernel, run NTP in slew mode (ntp -x) and have it active during leap second. (This is Google's leap smear method)

In MV QA testing, we have tested approach 1, which allows the NTP daemon to pass the leap second flag to the kernel which handles the leap second. The other two approaches (2 & 3) have not been tested.

In our testing, we have simulated by setting up a test NTP server and file from NIST. The test setup details are described in NTP support site [6]. The semi-automated test suite that we have designed and developed covers the entire testing of leap second through NTP. This would setup a local NTP server with leap second NIST file, which would simulate and propagate the leap second flag to all the NTP clients (target machines under test) and finally test if the NTP client in each target is able to eventually arm the CGE7 kernel with the leap second flag. Finally we test if the CGE7 kernel is able to insert the leap second flag and check for the following expected message in the kernel buffer (dmesg)

[XXX.XXX] Clock: inserting leap second 23:59:60 UTC

The test methodology includes putting the system under various stress before the leap second is inserted. The system is tested under tremendous stress using the following stress/load tools.

a. Futex Stress [5]

b. Realtime application load using cyclictest

c. Scheduler stress using hackbench

d. Filesystem stress using bonnie++

After the leap second is inserted by the CGE7 kernel we test for kernel deadlock, futex issues and for any high CPU load issues on the system/target under test.

Test Results:

None of the platforms showed any regression failures (kernel deadlock or high CPU load issues) noted in the past and on all platforms NTP daemon was able to successfully arm the CGE7 kernel with leap second flag and finally the kernel was able to successfully insert the leap second and showed the following expected message in kernel buffer (dmesg)

[XXX.XXX] Clock: inserting leap second 23:59:60 UTC

IV. MV Precision Time Protocol (PTP) Testing

Testing PTP for leap second issue is lot trickier and probably out of test scope due to two reasons.

1. Hardware time stamping is commonly implemented in PTP technology and hardware time stamping is implemented using special switches and routers called transparent clocks or boundary clocks. The test setup is lot more complex to deploy due to lack of such special switches/routers.

2. The PTPD package version used in MV CGE7 releases until April 2015 is ptpd-2.2.0 and unfortunately the appropriate code for handling leap second processing was introduced in v2.2.2 according to this announcement/release notes [7]

At the time of writing, MV engineering team has already updated the PTPD package to v2.3.1. MV Customers using PTPD for their timekeeping should contact MV Support for further details and update their package to PTPD v2.3.1 to handle the upcoming leap second.

Also there are simulation software [8] available to simulate and test PTPD leap second.

V. Test Suite

The leapsecond test suite developed by MV QA engineering team, to test both MV CGE7 kernel and NTP are semi-automated and available for use at https://github.com/premkarat/leapsecondtest

The README file has detailed instructions on executing the test.

VI. References

[1] https://www.meinberg.de/download/burnicki/Technical%20Aspects%20of%20Leap%20Second%20Propagation%20and%20Evaluation.pdf
[2] http://lwn.net/Articles/504744/
[3] https://github.com/johnstultz-work/timetests
[4] https://github.com/AmadeusITGroup/NTP-Proxy
[5] https://github.com/davidlohr/futex-stress
[6] https://support.ntp.org/bin/view/Dev/LeapSecondTest
[7] http://sourceforge.net/p/ptpd/mailman/message/29435153/
[8]https://www.meinbergglobal.com/english/products/mbgprotosim.htm