ISSN: 2393-8528

Contents lists available at www.ijicse.in

International Journal of Innovative Computer Science & Engineering

Volume 5 Issue 3; May-June 2018; Page No. 20-23

Using Real Time Linux Kernel with Computationally Intensive Workloads

Alexey Vasyukov[1], Alexey Ermakov[2]

[1] Moscow Institute of Physics and Technology,

Institutsky lane 9, Dolgoprudny, 141701, Russia

[2] Moscow Institute of Physics and Technology,

Institutsky lane 9, Dolgoprudny, 141701, Russia

ABSTRACT

This article is devoted to the problems of running in the cloud computing environment scientific and engineering applications, which require both a large amount of calculations and also a precise temporary resolution of operations. Such requirements typically arise in applications that receive data from the hardware, process it, and the results of processing are used to issue control commands for the equipment. The article considers the possibility of using a real-time Linux kernel to support this class of applications. Different hardware was tested, the results were obtained for achievable latency for real-time processes, and also the effect of the real-time kernel on the performance of the computing subsystem was measured (the number of GFlops in LINPACK tests, and the speed of working with memory on NUMA systems).

Key words: real time linux, soft real time, preempt_rt

I. REAL TIME LINUX KERNEL

The real-time system must ensure that the task that is being performed will receive CPU time at least after a specified interval. Therefore, the creation of real-time systems imposes very stringent requirements on low-level architecture. General-purpose systems (including generic Linux kernel) were initially designed without these requirements, so they cannot provide real-time operation. General-purpose systems focus on achieving the maximum overall system performance "on average". As a result, some applications may sometimes experience delays. In the vast majority of cases, a delay of a few tenths of a second will not be noticed, but there are tasks for which such times are critical.

For the Linux kernel, there is a set of patches PREEMPT_RT [1], in which the mechanisms of the work of the kernel with locks are revised, high-resolution timers are included, the non-preemtable parts of the kernel code are divided into minimal blocks, and a number of other optimizations are made. When using PREEMPT_RT, however, the required time for real-time task execution is provided on average, but may not be provided in the worst cases. In accordance with the generally accepted classification, the real-time Linux kernel should be referred to soft real-time operating systems.

However, for many tasks this level of determinism is acceptable. And in this case, another factor is important - unlike the specialized real-time operating systems, from the point of view of the API real-time Linux does not differ from the usual Linux kernel. As a result, it becomes possible to quickly transfer applications to the real-time operating system and start working with them in a new environment. Of course, this does not guarantee that required level of determinism will be achieved automatically at the level of application software. In any case, the process of mutual optimization of application settings, operating system and hardware infrastructure will be required. Nevertheless, it is extremely important that one does not need to rewrite the applications to get started. It is possible to start the system with the usual applications, measure the delays received at different stages of processing, and then make corrections only where they are needed.

Figure 1 shows the results for processing interrupts from equipment by a system based on a conventional Linux kernel. Three series of measurements are carried out, which are shown on the graphs in color. The processing time in nanoseconds is laid out horizontally; the probability density and probability are plotted vertically on two

Corresponding author: Alexey Vasyukov

graphs, respectively. Figure 2 shows similar results obtained by the system based on the real-time Linux kernel.

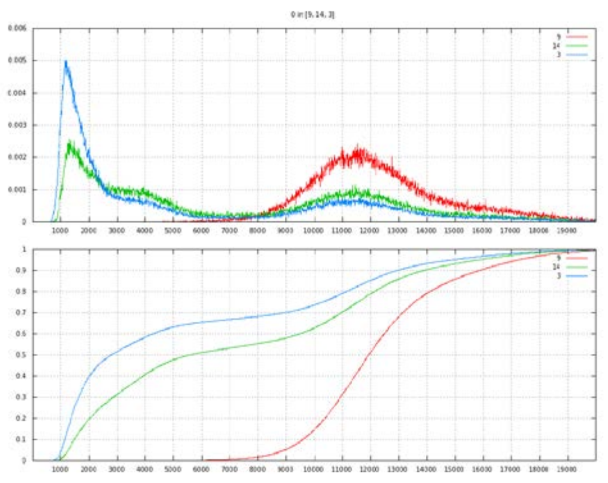

Fig.1. Interrupt handling, regular Linux kernel

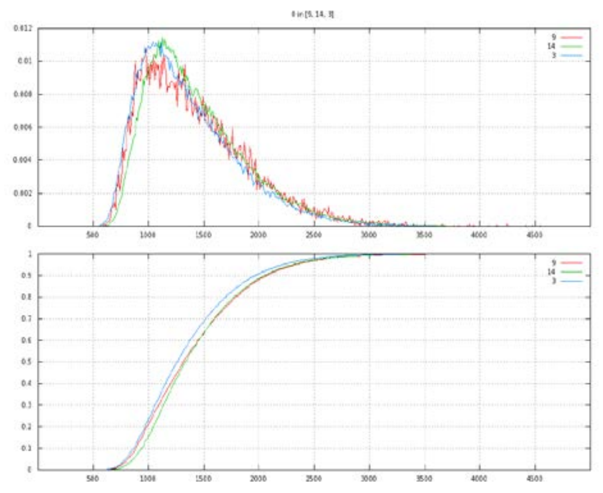

Fig.2. Interrupt handling, real-time Linux kernel

It can be seen that for a generic purpose Linux kernel, the results vary significantly for different series, and 90% of events fit within an interval of 0-16 µs. For the real-time kernel, the differences between series are insignificant, and 90% of the events fit within the 0-2 µs interval.

II. HARDWARE

Hewlett-Packard servers were used for testing, the hardware settings were performed in accordance with [2]. All machines run RHEL6.6 with a real-time kernel, kernel version 3.10.33-rt32.33.el6rt.x86_64. The characteristics of the processors of test machines are given in Table 1.

Table 1: Hardware specification

System	g9test1	g9test2	tks7	tks3
CPU type	Intel(R) Xeon(R) CPU E5-2695 v3	Intel(R) Xeon(R) CPU E5-2667 v3	Xeon(R) CPU E5-2690 v2	Intel(R) Xeon(R) CPU X5687
CPU Freq	2.30GHz	3.20GHz	3.00GHz	3.60GHz
CPU Cores	14	8	10	4
CPU Sockets	2	2	2	2
GFLOPS	1030	819	480	115.2

III. REAL TIME KERNEL AND HIGH CPU LOAD

The real-time kernel provides determinism, but it comes with some cost, significant amount of additional work is required in the kernel part of the system. This fact can be important for systems that perform complex calculations simultaneously with processing interrupts To assess the impact of the real-time kernel on performance, the standard Intel linpack 11.2.2.010 test [3] was used. The High Performance Linpack test is designed to evaluate the computing power of systems. The test consists of solutions of a system of linear equations. In the role of the result is the number of operations with a floating point per second (Flops). The theoretical performance values for all the systems considered in the tests are given in Table 1. It should be noted that the typical value of practical performance in HPC tests for modern systems based on the general-purpose Linux kernel is at the level of 90-95%.

Two HPL tests were conducted. In each test, the system performance was measured depending on the size of the problem (Fig. 3 and Fig. 4).

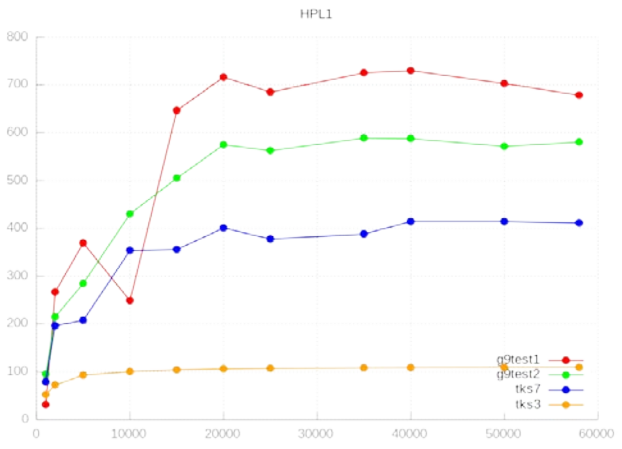

Fig.3: The performance of the systems in the Linpack test, the first series of measurements (X axis – problem size, Y axis – performance in GFLOPS).

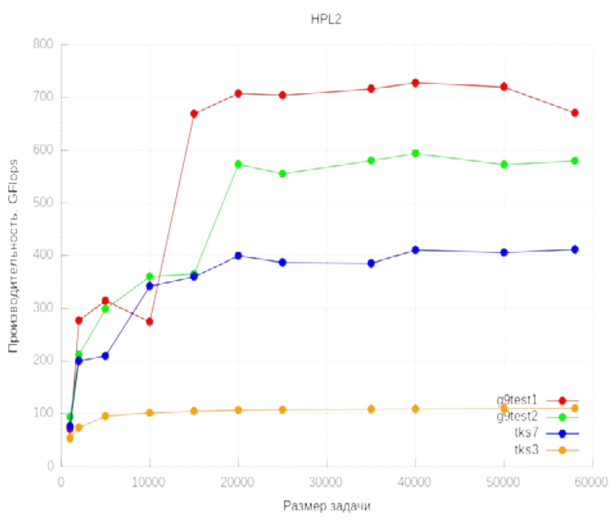

Fig.4: The performance of the systems in the Linpack test, the second series of measurements (X axis – problem size, Y axis – performance in GFLOPS).

Thus, the overhead costs when running an intensive computing load in a real-time environment depend on the equipment used, and range from 7% in the best cases to 32% in worst cases. Basically, degradation of performance is the expected result, but the value obtained clearly indicates that it is impossible to neglect it for resource-intensive applications.

To test RAM operations numademo utility [4] was used. We considered the tests numademo memcpy and numademo random2 (memory copy and random access). The block size in all tests is 256MB. The highest speed of work with memory can be achieved with calls within the same NUMA node. When calling "crossed" nodes, the minimum speed is achievable. Table 2 shows only the largest and smallest result for the average speed. These results do not differ from the results when using a general purpose kernel.

Table 2: RAM performance on NUMA system.

	g9test1	g9test2	tks7	tks3
Memcpy max	32119.49 MB/s	38936.98 MB/s	11545.11 MB/s	16696.45 MB/s
Memcpy min	23670.51 MB/s	24758.17 MB/s	6116.19 MB/s	9822.69 MB/s
Random max	143.32 MB/s	173.59 MB/s	182.87 MB/s	179.05 MB/s
Random min	122.74 MB/s	155.05 MB/s	158.91 MB/s	151.07 MB/s

To assess the achievable time resolution for real-time applications, the cyclictest utility [5] (version of the package rt-tests-0.83-1.el6rt.x86_64) was used. The cyclictest test allows to evaluate the realtime performance of the kernel. During the test, a thread is run on each core and wakes up on the timer. On waking, the thread measures the difference between the actual awakening time and the time when the thread should be woke up. This difference is the latency caused by operating system and hardware.

The test was started twice with the parameters "-U -H 500 -p 10". The first run was without the load for 5 minutes. The second run was for 20 minutes with Intel Linpack running simultaneously. Since the test was run with FIFO priority of 10, high background load should not have a significant effect on RT behavior.

The following figures show the latency distribution for the two test runs. Since the distributions have a clear maximum in the region of 0-10 µs, for convenience two graphs are presented for each test: the head of distribution and the tail.

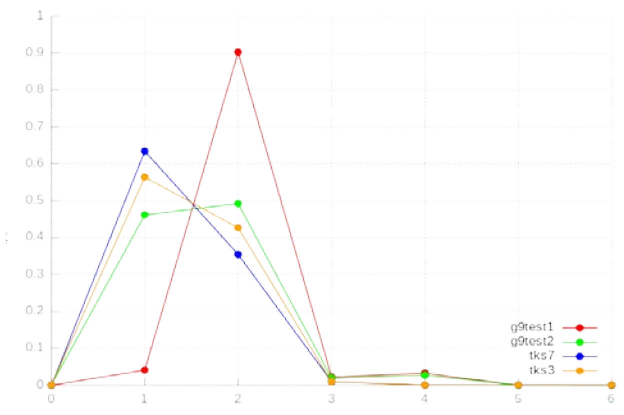

Fig.5: Test without load, head of distribution (X axis – latency in µs, Y axis – percentage of events with the given latency).

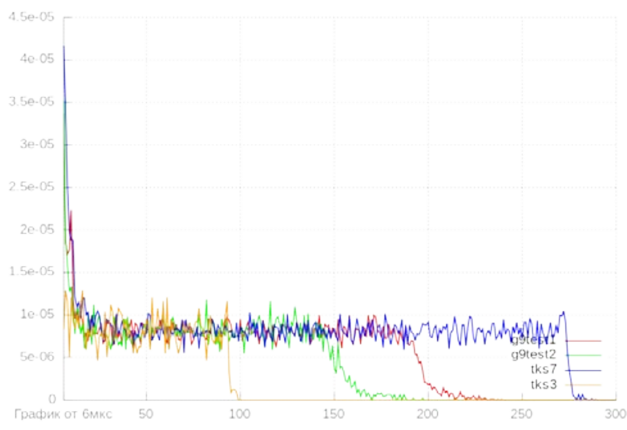

Fig.6. Test without load, tail of distribution (X axis – latency in µs, Y axis – percentage of events with the given latency).

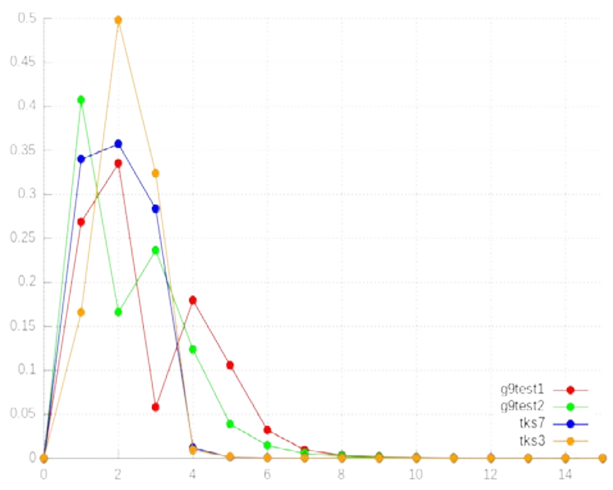

Fig.7: Test with HPL background load, head of distribution (X axis – latency in µs, Y axis – percentage of events with the given latency).

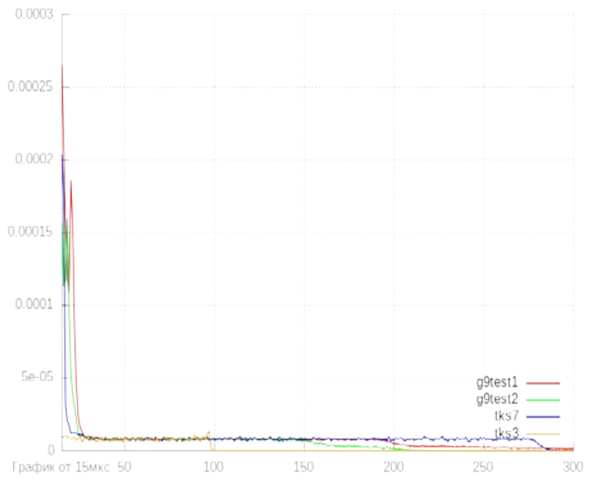

Fig.8: Test with HPL background load, tail of distribution (X axis – latency in µs, Y axis – percentage of events with the given latency).

Thus, when starting the real-time process without the load, 90% of the events fit into the 0-3µs interval. With a simultaneous operation of intensive computational load (HPL takes 100% of available processor time), 90% of events fit into the interval of 0-7µs. (For comparison, the general-purpose kernel gives a result of 0-16 microseconds without the load - see Fig. 2.). This result shows a sufficient quality of the real-time kernel (by the standards of soft real-time operating systems), including test cases with a heavy load on the system.

IV. CONCLUSION

The results show the possibility of using a real-time Linux kernel to support the operation of applications, which are characterized by both a large amount of computation and high requirements for the temporary resolution of operations.

The testing showed that achievable determinism for generic Linux kernel are the following: 90% of events fit within an interval of 0-16 µs, results differ significantly from test to test. For a real-time kernel, the differences between test runs are insignificant, and 90% of events fit within an interval of 0-3 µs when running without background load and in an interval of 0-7 µs when running simultaneously with HPL, that consumes 100% of the available CPU time.

The overhead of ensuring the given level of determinism by the real-time kernel depends on the equipment used. For tasks related to intensive computing, overheads vary from 7% in the best cases to 32% in the worst cases.

ACKNOWLEDGMENT:

The research was supported by Russian Foundation for Basic Research grant 15-29-07096.

REFERENCES:

1. Real-Time Linux [Online] Available https://rt.wiki.kernel.org/index.php/Main_Page [Accessed: June. 20, 2018]
2. Configuring and Tuning HP Servers for Low-Latency Applications [Online] Available http://www.fusionio.com/load/-media-/2ojjak/docsLibrary/Configuring_and_Tuning_HP_Servers_for_Low-Latency_Applications-c01804533.pdf [Accessed: June. 20, 2018]
3. Intel Linpack [Online] Available https://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download [Accessed: June. 20, 2018]
4. Libnuma toolkit [Online] Available http://oss.sgi.com/projects/libnuma/ [Accessed: June. 20, 2018]
5. Cyclictest toolkit [Online] Available https://rt.wiki.kernel.org/index.php/Cyclictest [Accessed: June. 20, 2018]