VIRTUALIZATION AND CPU WAIT TIMES IN A LINUX GUEST ENVIRONMENT

James F Brady
Capacity Planner for the State Of Nevada
jfbrady@doit.nv.gov

The virtualization environment presents the opportunity to better manage and more efficiently utilize computing resources. The resource sharing aspects of virtualization does, however, create some new challenges from an application implementation and system support perspective. This paper addresses one of those challenges; how does one monitor and manage the behavior of the virtual environment when contention for its physical resources causes significantly long delays in processing to occur? A specific example, with data, is provided where one CPU is shared among three Linux Guests and processing demand among these guests result in significantly long virtual CPU wait times.

1. Introduction

The virtualization environment provides the opportunity to better manage and more efficiently utilize computing resources. These efficiencies, however, create some new challenges for system support and application implementation personnel. This paper addresses one of these challenges; how does one monitor and manage the behavior of the virtual environment when contention for its physical resources causes significantly long delays in processing to occur?

This paper addresses the question by first describing the virtual environment and its key components. The discussion then focuses on analyzing some CPU wait times measured on a live system that are put into perspective through a series of questions and answers about virtualization and the potential causes of the long CPU wait times experienced. The paper ends with some concluding remarks and a set of recommendations for practitioners implementing applications in the virtual environment or supporting virtual platforms.

2. Virtual Environment

A virtual computing system consists of multiple virtual, "guest", operating systems sharing a set of physical resources that are controlled by a master operating system, sometimes called the "hypervisor". The basic idea is to more efficiently use and better manage computing resources by consolidating workloads from several physically separate systems into a single hardware environment.

This approach to increased computing resource efficiency has a lot of merit but is not issue free. It is similar to a medication focused on patient wellness which has side effects. The characteristics of these side effects and the magnitude of their impact varies from patient to patient, or in this case, from guest operating system to guest operating system and application workload to application workload. One such side effect, and the focus of this paper, is the possibility that at high resource utilization levels individual guests are put in a prolonged "wait state" queueing for physical resources. An example of such a wait state situation is illustrated in the tables, graphs and reports contained in the next section.

3. Wait State Example

The example under consideration consists of a single CPU IBM Z900 environment where VM is the hypervisor for several Linux Guests. The only significant workload occurring among these guests happens on Monday evenings at 5:00 PM when a scheduled, i.e., cron, job is started on three of them that creates a tar archive of each guest's file system and compresses it using the gzip program.

The VM environment in question is set to IBM RedBook™ specification and the NTP (National Time Protocol) daemon is running on each of the SUSE Linux Guests.

Table 1: Linux Guest Wait State Event Log

Description	Time Stamp
Warning: Sleep Time is **73 sec** but should be **60 sec**	Mon Jan 10 17:02:54 2005
Warning: Sleep Time is **71 sec** but should be **60 sec**	Mon Jan 10 19:20:06 2005
Warning: Sleep Time is **100 sec** but should be **60 sec**	Mon Jan 17 17:02:28 2005
Warning: Sleep Time is **78 sec** but should be **60 sec**	Mon Jan 17 19:24:48 2005
Warning: Sleep Time is **98 sec** but should be **60 sec**	Mon Jan 24 17:03:08 2005
Warning: Sleep Time is **115 sec** but should be **60 sec**	Mon Jan 24 19:19:05 2005

Table 1 above summarizes the wait state events observed during three weeks in January 2005 on one of the three guests.

These wait state events are captured by a Perl language script running on the guest (written by the author) that operates as follows.

The script saves an initial time stamp and calls the Linux sleep function with the sleep parameter set to 60 seconds. When the sleep time expires the script awakens and saves an ending time stamp. The difference between the two time stamps is computed and compared with the sleep interval requested. If the difference is greater than the tolerance value of 3 seconds a log record containing the information shown in Table 1 above is created. The sequence is looped through continuously. The concept behind this script is to check if the hypervisor honors the 60 second sleep call wakeup request within 3 seconds.

The script ran 24/7 during the three week period sampled and logged the six wait state events shown above. Linux Guest resource consumption statistics were also produced 24/7 during this three week period using data collection and analysis tools developed by the author. These tools invoke and process data gathered by standard Linux metering tools, i.e., vmstat and ps, the Sysstat library [SYS], i.e., iostat and sar, and Linux "/proc" system partition counters.

Below are bar charts produced by these analysis tools showing Linux Guest resource consumption for CPU, i.e., Figure 1, Disk I/O, i.e., Figure 2, and Network Packets, i.e., Figure 3, on Monday January 24, 2005 at hourly intervals between 1:00 PM and midnight. As Figure 1 depicts, CPU usage is nearly 100% for the 1700 hour and 1800 hour (5:00 PM through 7:00 PM) time periods while Disk I/O rates and Network Packet rates, Figure 2 and Figure 3, are significant but not excessively high.

Additionally, there is a bar chart and associated table identifying CPU consumption by process, i.e., Figure 4 and Figure 5. The running process list in Figure 4, labeled by name and ID number, indicates the two gzip processes account for nearly 100% of the CPU time consumed between 1600 hours and 2100 hours. Figure 5 contains the number of CPU seconds used by each process during the hours listed and identifies the tar and gzip CPU seconds for 1700 hours, 3245 seconds, to be 90% of that hour. Both Figure 4 and Figure 5 clearly show the gzip processes as the dominant contributor to CPU busy.

Finally, a partial listing of two hourly SAR (System Activity Report) reports, i.e., Figure 6.1, is shown that highlights two gaps in the 58 second sampling interval chosen. These gaps are identified with a "<---- Wait State" label. The first wait state event ends at 05:03:08 PM which is 9 seconds later than the 58 second sampling interval requested, i.e., 05:02:59 PM. Likewise, the second wait state event finished 35 seconds after its 58 second time interval i.e., 07:19:05 PM instead of 07:18:30 PM. It is interesting to note that these two SAR report time stamps, 05:03:08 PM and 07:19:05 PM, have the same value as the two January 24, 2005 Wait State Event Log time stamps listed in Table 1 and repeated in Figure 6.2. This time stamp equality between two independently running processes is clear evidence that wait state events occurred and their occurrence is correlated with contention for CPU resources.

The resource consumption patterns for Monday January 10, 2005 and Monday January 17, 2005 are not displayed in this document but have similar characteristics to those shown for January 24, 2005.

Figure 1: CPU Busy

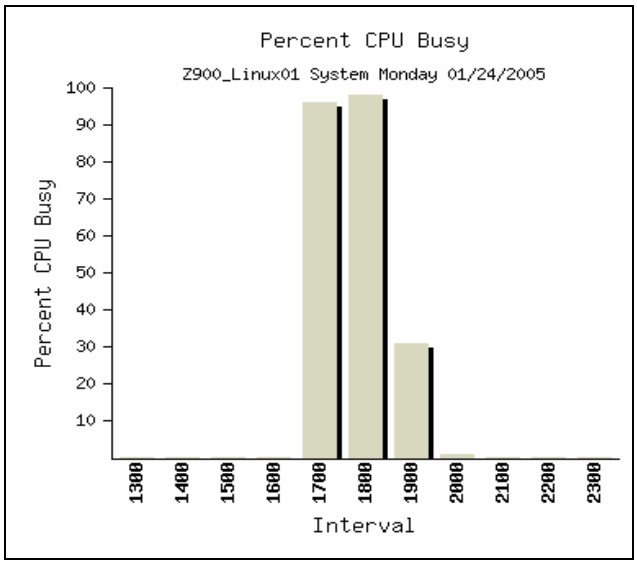

Figure 2: Disk I/Os per Second

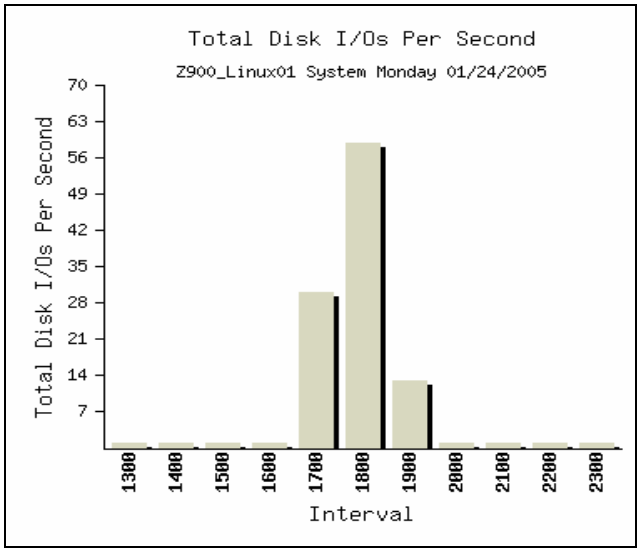

Figure 3: Network Packets per Second

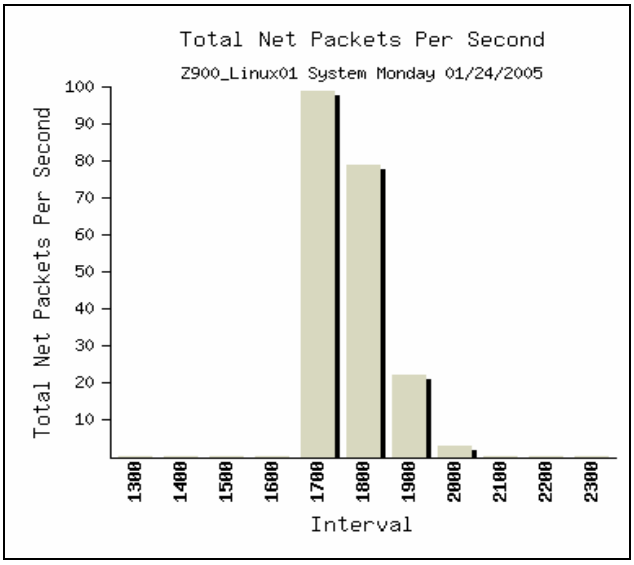

Figure 4: CPU Process Percentages

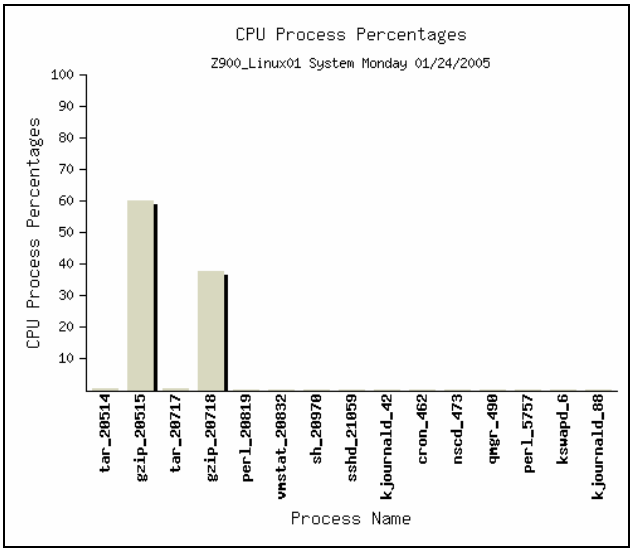

Figure 5: CPU Seconds Used By Each Process Hourly

Process Status Summary Statistics - Z900_Linux01 System Monday 01/24/2005

		--1600---		--1700---		--1800---		--1900---		--2000---		----Sum-----	
name	pid	sec	%	sec	%	sec	%	sec	%	sec	%	sec	%
tar	20514	0	0	34	1	0	0	0	0	0	0	34	0.64
*** **gzip**	**20515**	**0**	**0**	**3207**	**99**	**0**	**0**	**0**	**0**	**0**	**0**	**3207**	**60.27**
tar	20717	0	0	0	0	30	1	0	0	0	0	30	0.56
*** **gzip**	**20718**	**0**	**0**	**0**	**0**	**2009**	**98**	**0**	**0**	**0**	**0**	**2009**	**37.76**
perl	20819	0	0	0	0	0	0	2	15	0	0	2	0.04
vmstat	20832	0	0	0	0	0	0	1	8	0	0	1	0.02
sh	20970	0	0	0	0	0	0	1	8	0	0	1	0.02
sshd	21059	0	0	0	0	0	0	0	0	2	29	2	0.04
kjournald	42	0	0	1	0	1	0	2	15	3	43	13	0.24
cron	462	0	0	0	0	0	0	1	8	0	0	1	0.02
nscd	473	0	0	0	0	1	0	0	0	0	0	1	0.02
qmgr	490	0	0	0	0	0	0	1	8	1	14	2	0.04
perl	5757	0	0	0	0	0	0	0	0	1	14	2	0.04
kswapd	6	0	0	3	0	7	0	5	38	0	0	15	0.28
kjournald	88	0	0	1	0	0	0	0	0	0	0	1	0.02
	total	0	0	3246	100	2048	99	13	100	7	100	5321	100.00

***** Indicates high CPU usage**

Figure 6.1: System Activity Report (SAR)

```
Linux 2.4.21-83-default (Z900_Linux01)       01/24/2005

05:00:05 PM       CPU     %user     %nice   %system     %idle
05:01:03 PM       all      0.62      0.00     63.98     35.40
05:02:01 PM       all      0.33      0.00     39.10     60.57
05:03:08 PM       all      0.12      0.00     92.88      7.00 <---- Wait State
05:04:06 PM       all     94.81      0.00      2.90      2.29
05:05:04 PM       all     61.19      0.00     21.43     17.38
05:06:02 PM       all     95.02      0.00      4.40      0.59
05:07:00 PM       all     69.02      0.00     15.83     15.16
05:07:58 PM       all     87.28      0.00      7.58      5.14
05:08:56 PM       all     83.71      0.00      9.69      6.60
05:09:54 PM       all     91.33      0.00      4.59      4.09
05:10:52 PM       all     74.84      0.00     11.00     14.16
05:11:50 PM       all     88.09      0.00      5.52      6.40
05:12:48 PM       all     96.71      0.00      1.71      1.59
05:13:46 PM       all     86.14      0.00     12.31      1.55
05:14:44 PM       all     93.40      0.00      5.78      0.83
    :              :         :         :         :         :
07:13:40 PM       all     93.24      0.00      3.41      3.34
07:14:38 PM       all     94.66      0.00      2.95      2.40
07:15:36 PM       all     95.29      0.00      2.86      1.84
07:16:34 PM       all     93.97      0.00      2.74      3.29
07:17:32 PM       all     73.43      0.00     22.33      4.24
07:19:05 PM       all      2.31      0.00     97.39      0.30 <---- Wait State
07:20:03 PM       all      2.07      0.00      8.98     88.95
07:21:01 PM       all      0.21      0.00      0.36     99.43
```

Figure 6.2: Wait State Event Log

```
Warning: Sleep Time is 98 sec but should be 60 sec - Mon Jan 24 17:03:08 2005
Warning: Sleep Time is 115 sec but should be 60 sec - Mon Jan 24 19:19:05 2005
```

Note: The SAR report is in 12 hour time and the Wait State Event Log is in 24 hour time.

4. Perspective

What insights can be gained from this virtual environment wait state example? First, the situation can likely be mitigated by changing the schedule of the cron job and staggering the backup start times. This action was taken and the wait state events no longer occur.

This procedure fixes the immediate problem but looking beyond the basic "fix it" perspective and taking a more conceptual approach to the situation can be instructive. Such an approach gives rise to the questions contained in the next section regarding the virtual environment and how it differs from the standalone Linux system situation.

5. Virtualization Questions

When one reads the literature, some of which is listed in the references, they discover that virtualization is much more complicated to accomplish that it appears to be on the surface. The following set of questions address virtualization issues such as time references, hypervisor visibility into the guest workloads, and traffic congestion behavior.

Would the "Wait State Event Log" entries have occurred in the standalone Linux environment? The answer, with near certainty, is no. The reason for this assertion is a Linux system in standalone mode possesses a time share scheduler that runs every 10 milliseconds and dynamically assigns process priorities on a resource consumption basis. The scheduler is likely to give a relatively higher priority to the Wait State Event Log script and a lower priority to the gzip processes because the script only wants to run every 60 seconds for a very sort period of time and the gzip processes ask to run continuously for up to an hour.

How much visibility does the hypervisor have into the characteristics of each Linux Guest's workload? Given the Wait State Event Log results shown in the example above, the answer appears to be, not enough visibility. This is a situation where three Linux Guests that have identical workloads want the single physical CPU continuously for a long period of time. The hypervisor operates at the Linux Guest level not the process level within each Linux Guest, making CPU execution priority assignments difficult.

What effect does the hypervisor's apparent lack of workload visibility or potential lack of physical resources have on the applications running under the virtual umbrella? This is a complex "it depends" answer. The gzip process, which is essentially a "batch" oriented task, is affected very little by a wait state event and simply takes a little bit more time to complete its work. Transaction oriented processes or

processes which rely on “I’m alive” interaction, such as data replication functionality, can potentially fail, at least temporarily, as the result of a wait state event. At a minimum, transaction processing response times may become erratic and one or both sides of a data replication application could be put in standalone mode.

Also, as illustrated in the Figure 6.1 SAR report, metering packages in the Linux Guest can be rendered unreliable when running under the hypervisor. The SAR data produced for the two wait state entries in Figure 6.1 show more than 90% kernel (%system) utilization, which is suspicious when compared to values in the other sample entries. See [VEL05] at the end of this document for further discussion of this metering package issue.

What exactly caused the “Wait States” in the above example to occur? The cause appears to be contention for CPU resources but it is possible there is some issue with the Disk I/O, Network Packet, or Memory interfaces. The important thing is to recognize this as a virtualization issue and not a bug in either an application or a third party software product such as a DBMS (Data Base Management System). A tool like the Wait State Event Log Perl script can be a great aid in isolating virtualization oriented anomalies from those attributable to applications or third party software packages.

How does traffic congestion oriented resource degradation compare between the virtualization and Linux standalone environments? The answer to this question assumes that in both environments applications are well designed, i.e., contain no deadlock states and have properly sized system resources, e.g., message queues. In the standalone Linux case, increases in traffic volume cause system response time to grow systematically, i.e., the system slows down as traffic goes up. In the virtualization environment, however, high resource utilization can lead to long wait states that may cause timeouts to occur making application failure possible.

What other virtualization “side effects” exist but are not illustrated in the example being discussed? The example discussed in this paper is about as basic as it gets. There are three plain SUSE Linux Guests running tar/gzip processes for backup purposes at the same time.

One Linux modification very prominent in the literature, [IBM], that has the potential to emit some side effects is to **change the Linux 10 millisecond interrupt interval, sometimes called the 100 Hertz “jiffy” timer to a lower frequency or make it on-demand oriented.** The jiffy timer controls the Linux operating system interrupt interval which is the interval between scheduler invocations when priorities are dynamically assigned and processes put on the ready to run list. The reason this change is desirable from a virtualization perspective is the hypervisor needs to handle the interrupts even when the Linux Guest has no work to accomplish. If the Linux Guest operating system is modified to generate interrupts less frequently, or better yet, on demand, the hypervisor can manage its physical resources more efficiently.

Making this interval variable will reduce hypervisor overhead at low traffic levels but if there is a lot of traffic demand among multiple Linux Guests, it will likely not help much. There is also the possibility that such timing manipulations will create destabilizing timing complexities between Linux and its virtual peripheral devices, e.g., communications interfaces.

Another issue being mentioned in the literature is **memory synchronization associated with virtualized SMP (Symmetric Multiprocessing) environments.** Memory synchronization in SMP environments is generally accomplished using locks such as mutex (Mutually Exclusive Context Switch Locking) or futex (Fast User Space Locking). SMP inefficiencies occur in the virtual environment if one virtual processor of a multiprocessor domain is preempted when holding a lock, while other virtual processors of the same domain continue to run on other processors, waiting for the lock to be released. For a discussion of this issue see [XENc].

6. Conclusions

The virtualization environment creates an opportunity to better manage and more efficiently utilize computing resources. The efficiencies gained, however, are offset to some degree by the challenges and issues just discussed.

Fundamentally, virtualization takes each guest operating system out of the native environment for which it has been tuned and places it under a controlling umbrella. Linux, for example, is a very transaction oriented operating system which implements a time share scheduler that performs dynamic process priority assignment based on resource consumption characteristics at 10 milliseconds intervals. Any significant alterations to this operating environment imposed by the hypervisor could negatively impact Linux performance during high traffic periods.

The major virtualization concern, from a business perspective, is that workloads are aggregated for several underutilized standalone servers into a single virtual environment which, over time, becomes traffic congested and exhibits poor performance degradation characteristics. These performance degradation characteristics can range from erratic response time to

application instability and are much less desirable than the gradual system slow down generally experienced under Linux standalone conditions.

A major difficulty with discussing virtualization issues is that a large number of them surface when, as illustrated in this paper, resources are being heavily utilized and virtualization is operating at its highest efficiency. Comments from current virtualization customers not experiencing these issues should be put into perspective because those customers may be operating at minimum traffic volumes, with straightforward applications, and very low virtual/ physical resource ratios.

7. Recommendations

In light of the wait state example provided, the virtualization issues presented, and conclusions drawn, the following virtualization implementation recommendations are offered for consideration.

1. Implement a wait state event logging monitor on each guest, like the Perl script described in this paper, to help isolate virtualization issues from hardware, third party software, or application faults. The best monitor sleep parameter and tolerance level to use depends on the time criticality of applications. Smaller settings do a better job of identifying wait states but they create more overhead.

2. Monitor system resource consumption in each guest at the process level to help correlate wait state events with their cause. Remember that metering package samples taken during wait state events are suspect so cross-check them with the wait state event log and value the affected metering package samples accordingly.

3. Analyze the system resource consumption characteristics of any application being implemented in the virtual environment for the possible impact of wait state events. For example, would a ten second wait state event disable data replication between this application instance and its geographically separated paired instance?

4. If the application is designed to run in an SMP environment and uses memory synchronization locks like mutex or futex locks, be sure the virtualized SMP environment possesses lock status capabilities.

5. Check any scheduled activity the application performs to be sure it's schedule does not conflict with those of other guests. Removing the synchronous timing of the cron job that performs backups for the three Linux Guests mitigated the wait state situation discussed in this paper.

6. Check the ratio of virtual resource counts to physical resource quantities. The higher the ratio the greater the chances of resource contention and the more likely long wait state events will occur. Virtualization product vendor guidelines that take into account anticipated traffic volumes and application resource consumption characteristics are required for this analysis.

7. Be sure to traffic capacity test applications being installed on the virtual system to insure they meet performance expectations and integrate well with other virtual system applications. This traffic testing applies to both new applications and applications being ported from standalone platforms.

8. Summary

This document discusses the behavior of the virtual environment when resource utilization is high and contention for these resources is great. It contains a description of the potential side effects of this contentious situation and supports the assertions made with real world data. An analysis of the data provides the basis for a list of issues raised from a conceptual perspective about implementing applications in the virtual environment. Conclusions are drawn from the issue discussion and implementation recommendations are made as a result.

9. References

[IBM] "IBM Performance Considerations for Linux Guests", www.vm.ibm.com/perf/tips/linuxper.html .

[SMI05] Phil Smith III "Help! My (Virtual) Penguin is Sick!", IBM Z Journal, December 2004/ January 2005, www.zjournal.com/Article.asp?ArticleID=971

[SYS] Sebastien Godard "The Sysstat library" Linux metering package.
http://perso.wanadoo.fr/sebastien.godard

[VEL05] "Linux on VM", Velocity Software Inc, www.linuxvm.com

[XENa] "Interface manual", The Xen Team, University of Cambridge, UK, 2002-2004, www.cl.cam.ac.uk/Research/SRG/netos/xen/

[XENb] "User's manual", The Xen Team, University of Cambridge, UK, 2002-2004, www.cl.cam.ac.uk/Research/SRG/netos/xen/

[XENc] "SMP Guest Proposal", xen-devel archives, April 2005, http://lists.xensource.com