# Understanding Linux on z/VM Steal Time

June 2014

**Rob van der Heij**

*rvdheij@velocitysoftware.com*

## Summary

Ever since Linux distributions started to report “steal time” in various tools, it has been causing grief and confusion among Linux experts with a background in dedicated server configurations. Extreme values of steal time certainly justify further investigation, like any extreme values for performance metrics should do. While it is true that resource contention may show as steal time, it is too simple to consider any non-zero steal time as evidence of damage and halt diagnostics while claiming a “hypervisor problem.”

This paper explains the factors that make up steal time as perceived by Linux, and introduces the zVPS screens and reports that help to explain and sometimes predict steal time.

## Conclusion

In a shared resource environment, there is always a chance that a server must wait for its resources. Some amount of latency is unavoidable. This is the consequence of sharing a resource. It is unrealistic to expect no latency in access to a shared resource. A high average utilization leads to higher average latency, but large configurations with many CPUs can run at higher utilization without excessive increase in latency.

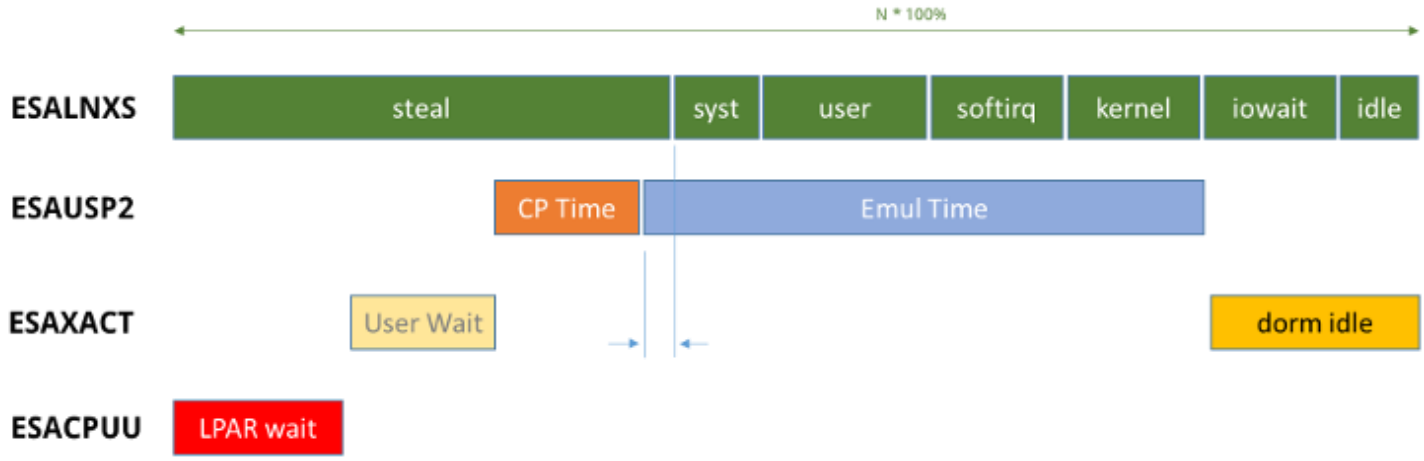
With Linux on z/VM, the CPU resources are shared both at virtual machine level and at LPAR level. At both levels, over-commit of resources leads to contention and will cause latency in getting the CPU resources. Metrics in zVPS reports are needed to determine whether reported steal time is a true concern and would need to be addressed.

Because some of the metrics are collected by state sampling and system wide averages, steal time can only be predicted with limited accuracy. The numbers can be used however to provide a range of steal time percentage that is reasonable for a given configuration and utilization.

The focus on steal time may still be misleading. The real question is whether the service levels for the applications can be met by the platform. When Linux on z/VM can deliver response times using a shared fast CPU and some wait time, then that could be an adequate solution for the business. The desire to have a fast CPU and not having to share it with others is understandable, but may not be attractive from a business point of view.

## Introduction

The diagram below is used to introduce the various components of time from the Linux guest perspective, and explain the relation with metrics that represent CPU resources in the system.




The following sections show various zVPS displays during execution of a performance experiment.

### ESALNXS – Linux Performance View

The top section in the chart represents the Linux components of time.

| | |
|---|---|
| user | CPU usage to run the actual application program |
| syst | System calls issued by the program (eg file system operations, creating processes, etc) |
| softirq | Second level interrupt handler (eg network I/O) |
| kernel | Interrupt handling (eg disk I/O) |
| iowait | Waiting for disk I/O to complete |
| idle | No work to do |
| steal | Anything not covered by the other components |

These metrics can be found for example on ESALNXS for the virtual server. ESALNXS here shows[1] a total of 18% consisting of 9.5% attributed to processes (2.6% plus 6.9%) and 8.4% for Linux overhead (0.7% plus 7.7%). Together with the I/O wait of 74.1% this represents 92% of the time. The remaining 8.0% is reported as steal time (this is how Linux computes it).

```
ESALNXS 2/2           <-- Processor Pct Util -> <CPU% Overhead>  I/O
Time     Node     CPU Total User Nice Syst Idle Krnl  IRQ Steal Wait
-------- -------- --- ----- ---- ---- ---- ---- ---- ---- ----- ----
08:05:00 roblx1   Tot  18.0  2.6    0  6.9    0  0.7  7.7   8.0 74.1
```

[1] The examples were done with “prorate” disabled in ZWRITE. When prorate is enabled, Linux overhead is distributed proportionally over the processes. This is useful for charge back purposes, but makes it a bit harder to match the various numbers as we want to do here.



Velocity Software, Inc. 196-D Castro Street, Mountain View, CA 94041-1202
T: +1 650 964 8867 | +1 877 964 8867 | F: +1 650 964 9012 | www.velocitysoftware.com

Most Linux tools normalize CPU usage to 100%. The zVPS percentages are related to a full CPU. In this test the guest was using only one virtual processor, the total in ESALNXS is N x 100% (N is the number of virtual processors).

### ESAUSP2 – z/VM User Performance View

The next layer in the chart represents the CPU usage of the guest as measured by z/VM, and reported in ESAUSP2. The "iowait" and "idle" portions do not represent CPU usage, so during those periods the virtual processors of the guest do not run. From a z/VM point of view, all Linux CPU usage (user, system and overhead) is "Emulation Time" where the guest is running on the real CPU undisturbed at full speed.

```
ESAUSP2 0/3       <Processor>
         UserID   <-use CPU%>
Time     /Class   Total  Virt
-------- -------- ----- -----
08:05:00 ROBLX1   19.22 18.59
```

ESAUSP2 for this interval shows 18.6% Emulation Time ("Virt") which corresponds to the 18% that Linux reports in ESALNXS. The difference can be explained by the code Linux has to run before it can actually measure time.[2] Granularity of time as reported here (in units of 10 ms) may cause some skew in reporting.

When looking at 1-minute intervals and fairly short peaks of activity, some discrepancy between Linux and z/VM metrics can be noticed because the data collection intervals are slightly shifted to ensure that Linux data arrives early enough to be picked up for the monitor interval. The effects average out over time, but can be confusing when studying data for short periods.

The difference between "Total" and "Virt" (0.6%) is the CP Time component. This CPU usage not visible as such in Linux. It represents z/VM functions on behalf of the workload in this guest. It is sometimes called virtualization overhead because part of it is to isolate guests from each other and from the actual hardware. However, part of the CP work is for functions on behalf of the guest so the guest does not have to do it. For example when the guest is using a VNIC connected to a VSWITCH, Linux CPU usage is less than when the guest would be using dedicated OSA devices. The CPU usage by CP is both for virtualization of the resource and for working with the real hardware devices.

### ESAXACT – z/VM User Wait States

The third layer represents metrics that explain why a virtual CPU is not running. The most obvious reason would be when the virtual server has no work to do for the virtual processor. It could be (briefly)[3] waiting for an I/O operation to complete, or waiting for a network data or for a timer to expire. The other reason is "User Wait" which means the guest has dispatched work on its virtual CPU and would like it to run, but for some reason it currently does not run. Those reasons are shown on ESAXACT. The metrics in ESAXACT are based on state sampling by CP. These 1-second samples do not provide the same accuracy as we get from

[2] Modern CPUs rely on various levels of cache around the CPU to run at clock speed. It is fair to assume that when a virtual CPU is dispatched by z/VM, these first instructions may take longer because the instructions are not yet in the CPU cache. The metrics and tuning options in this area deserve a research paper of their own.
[3] The difference between idle and dormant is based on how long the virtual CPU is not using any resources. Many applications and middleware components use "polling" techniques. This means that a Linux guest with no actual workload will still issue very frequent timer requests and never be true idle according to z/VM.



Velocity Software, Inc. 196-D Castro Street, Mountain View, CA 94041-1202
T: +1 650 964 8867 | +1 877 964 8867 | F: +1 650 964 9012 | www.velocitysoftware.com

the CPU timer metrics with most state changes happening 10,000 times more often. Although it is possible to sample virtual CPU state more often, the rough numbers are typically enough to understand what is happening in the system.

```
ESAXACT 2/2       <-Samples-> <---Percent non-dormant------->
         UserID          Pct               E-  T-     Tst      D- <---CPU%--->
Time     /Class   Total  In Q Run Sim CPU SVM SVM  CF Idl Oth SVM Assign  Used
-------- -------- ----- ----- --- --- --- --- --- --- --- --- --- ------ -----
08:05:00 ROBLX1     120  50.0  20   0  10   0   0   0  70   0   0    3.8  19.2
```

In this case ESAXACT shows the virtual server actually has 2 virtual CPUs (120 samples per minute). Since the virtual CPUs were in queue 50% of the time, we must conclude that one virtual CPU was dormant (it was actually varied offline in Linux). The other percentages break down the non-dormant portion again. In this case 20% of the time the processor was running, 10% waiting for CPU (competing with other virtual machines to be dispatched on one of the logical CPUs of z/VM) and 70% “test idle” wait. This represents brief periods where the guest did not have work to do, but after a very short time needed CPU resources again (eg waiting for a disk I/O to complete, or waiting for a timer interrupt).

An alternative presentation of the state samples is shown below. This screen does not normalize the wait states by non-dormant portion and makes some numbers more obvious when virtual processors are not dormant all the time.

```
ESAWAIT4 0/1      <--------- Virtual CPU State Percentage --------->  Poll
Time     User       Run CPUwt  CPwt Limit  IOwt PAGwt Othr Idle Dorm  Rate CPU% ms/tx
-------- -------- ----- ----- ----- ----- ----- ----- ---- ---- ---- ----- ---- -----
08:05:00 ROBLX1    20.0  10.0     0     0     0     0    0 70.0  100 622.3 19.2  0.31
```

An interesting metric here is the “poll rate” which shows the number of times per second the guest went idle and woke up again. The 622 here means that on average 622 times per second the guest needs to get in line to be dispatched and consume some CPU cycles (0.31 ms on average). Even the slightest latency will add up to something significant when you encounter that 622 times per second. From the 10% CPU wait above, we can conclude that the average latency is around 0.15 ms (100 ms / 622).

### ESACPUU – LPAR Wait State

Many installations also share the real CPUs among multiple LPARs. This means that when z/VM dispatches work on one of the logical CPUs of the LPAR, there will be some delay before PR/SM dispatches the logical CPU on a real CPU. This “involuntary wait” or “LPAR steal” shows on ESACPUU.

```
ESACPUU 1/3       <----CPU (percentages)----------> <External (per second)>
         <--CPU-> Total  Emul <-Overhd>  <CPU Wait> <--Page---> <--Spool-->
Time     Type  ID  util  time User Syst  Idle Steal  Read Write  Read Write
-------- ---- --- ----- ----- ---- ---- ----- ----- ----- ----- ----- -----
08:05:00 CPU  Tot  55.9  46.9  5.8  3.3 131.0  13.0     0     0     0     0
```

The total LPAR Steal here is 13%. When the LPAR waits 13% to consume 55.9% of CPU, we divide the two numbers to determine that it takes about 25% extra time for the system to get the work done. Because



Velocity Software, Inc. 196-D Castro Street, Mountain View, CA 94041-1202
T: +1 650 964 8867 | +1 877 964 8867 | F: +1 650 964 9012 | www.velocitysoftware.com

the logical CPUs are still idle 130% of the time, the system does eventually get its resources and the LPAR does not starve.

### Estimating Steal Time

Using the metrics from the different levels of virtualization and resource sharing, we can try to explain steal time observed in Linux for the sample data shown above.

On the Linux level, the system can account 18% of CPU while z/VM actually spends 19.2% of CPU for the guest (ESAUSP2). This difference already explains an elongation of 1.1, which corresponds to a steal time of 1.2% measured by Linux.

From the LPAR data, we know that contention on LPAR level will on average delay CPU usage by a factor 1.23 in this case. Lacking any other information, our best guess is to assume our server is affected in the same way. That means the 19.2% CPU (the usage as determined by the z/VM metrics) will take 1.23 times as long, so 23.6% of time. The difference of 4.4% would be reported as steal time by Linux.

The final component is the delay caused by CPU wait in z/VM. The ESAXACT data in this case suggests an elongation of 1.5 but this is really a worst case estimate. The reason is that the Linux system is polling[4] which makes the server queue for CPU resources very often when idle, even when it doesn't really need CPU resources. When the Linux system is not idle, these timer requests are handled during the time slice itself and do not cause any CPU wait.

Based on these factors, we conclude that it would take between 23.6% and 35.4% of time to deliver the CPU resources for the guest. This means the steal time measured by Linux would be between 5.6% and 17.4%. The ESALNXS data shows 8% which is well within this wide range.

When we repeat the measurements with a workload that is entirely CPU constrained (so does not leave any idle time with polling) the wait state samples are more accurate and we can determine steal time better. A workload that depends entirely on CPU resources is also more affected by steal time than a workload that is waiting for I/O most of the time.

### Implications of Steal Time

Although a latency of a factor 1.4 may sound dramatic, it is more important to understand the impact of that latency on the throughput of the server. The mixed workload is waiting much more on I/O than on CPU, so even if we had dedicated CPU resources for the workload, the performance gain would be at most 8%. Considering that System z CPUs are typically faster than other platforms, it might still be attractive to accept 8% latency for getting the CPU resources in a shared resource environment. In a real business environment, what matters is to meet the required response time at lowest cost.

Concern about latency without looking at processor speed is like complaining about boarding times for a long flight: certainly if you would walk the way you could start right now rather than wait for an hour. But the actual response time of the transaction is likely much worse.

[4] In addition to the I/O test program used to generate this workload, the Linux guest is also running a popular database server which keeps track of time using a 1 ms timer interrupt even when the system is idle. This is a rather unfortunate design choice to address issues that have been resolved already in current Linux kernels.



Velocity Software, Inc. 196-D Castro Street, Mountain View, CA 94041-1202
T: +1 650 964 8867 | +1 877 964 8867 | F: +1 650 964 9012 | www.velocitysoftware.com