Why Relative Share Does Not Work

Velocity Software, Inc

March 2010

Rob van der Heij

rvdheij @ velocitysoftware.com

Introduction

Installations that run their production and development Linux servers on the same z/VM LPAR often use the relative share setting to favor their production workload over development. While a higher relative share indeed gives a virtual machine access to more CPU resources, the amount of CPU resources obtained that way appears disproportional to the share setting. It is often observed that the virtual machine with lower share setting get far less than the intended share of resources.

Conclusion

Linux on z/VM workloads differ significant from the type of workload the z/VM scheduler was originally designed for. Linux servers often show more or less “polling” behavior that confuses the scheduling algorithms in z/VM. Each idle Linux guest creates excess share. When that accumulates to a large portion of total resources, the customer is unable to distribute CPU resources as the business requires. Relative share still works, but it controls only part of the CPU resources. When it controls only a tiny part of the resources, to most of us it appears not to work.

This problem can be mitigated by proper share settings (including the correction of some bad default settings). Further resolution can be achieved by additional tooling on Linux and z/VM. The ESALPS components for this are under development.

An alternative approach is to use the number of virtual CPUs (all with same per-CPU relative share) as the sole mechanism to allocate CPU resources. At the expense of some additional overhead on z/VM and Linux, this may provide a useful option to allocate CPU resources. The granularity of this approach is rather limited, and less useful for small installations.

Relative Share and the z/VM Scheduler

The share setting of a virtual machine is the main control to influence the z/VM scheduler in distributing CPU resources when there is a CPU constraint. When demand for CPU resources is larger than available resources, virtual machines will have to wait to get there share of the CPU. The share setting is used to make some virtual machines wait longer than others. That way they get a less CPU resources.

To simplify the discussion, we ignore the option to assign absolute share, and focus on relative share for a moment. With relative share, we express the importance of a virtual machine in relation to the others. The idea is that when two virtual machines compete for resources with relative share 100 and 200 respectively, the CPU resources are distributed 100:200. Or stated differently; when no other virtual machines are competing, one gets 33% and the other 66% of the available resources.

The consequence of this is that the value of REL 100 (the default share setting) varies with the demand for CPU resources. At any given time, one can add up the relative share of all virtual machines in queue and divide the available CPU resources by that number. This gives the amount of CPU resources that correspond to a certain relative share. For example, imagine a z/VM system with 2 IFLs and assume 10 virtual machines with REL 100 are in queue. The total relative share is 1,000 for 200% of CPU, so REL 100 corresponds to 20% of CPU.

The z/VM Scheduler

The way the z/VM scheduler gives virtual machine more or less access to CPU resources is through the delay with which they get their time slice. This is an indirect approach to control CPU resources, but unlike other algorithms it avoids starvation of virtual machines and ensures that each virtual machine makes some amount of progress.

By observing the in-queue virtual CPUs and their relative share, the scheduler can predict the delay each virtual CPU will observe to get its next time slice, given the overall demand for CPU (the in-queue virtual CPUs) and their relative share. This translates into a deadline by which the virtual CPU is entitled to have its time slice.

Because an idle Linux server still has some background “noise” going on, its virtual CPUs are frequently in-queue to consume a tiny amount of CPU cycles. Unlike traditional workload that consumes full time slices until it blocks on I/O or goes dormant, the Linux virtual CPUs get in line each time again. When the requests for CPU resources are frequent enough, their virtual CPU is mostly in “Test Idle” state.

For example, imagine a Linux server with the 100 Hz timer ticking[1] using 0.5% of a CPU. This translates to 5 ms of CPU per second (actually, one time slice) but consumed in many small pieces. In theory one would expect 100 times 50 µs of CPU. In practice, the dispatching latency in z/VM prevents this, so the virtual CPU is probably serviced 10 times a second, each time using 0.5 ms of CPU, or one tenth of a time slice.

With many Linux servers getting in queue all the time and consume just a small portion of a time slice, the scheduler’s assumptions are too pessimistic for the CPU contention on the system. The computed deadline for virtual machines to get their CPU resources is set to far in the future. This means that frequently the next virtual CPU in line is not yet near its deadline and the dispatcher considers it has spare time available. This then allows virtual machines with higher relative share can get ahead of the line and consume another time slice. On systems with a lot of idle Linux virtual machines, it is not uncommon to find 80% of all CPU resources to fall in this spare time category. This would allow the virtual machine with a slightly higher relative share setting to use a large amount of the CPU resources.

Experiment

The problem is that users get in queue even when they have no real work to do. This confuses the z/VM scheduler because it is expecting a lot of work. With REL 10,000 it is obvious that things get messed up, but even a few Linux virtual machines with REL 300 reveal the problem.

We start our experiment with 3 looping users REL 100. They all get equal share of the resources and this is as we would expect.

```
Screen: ESAUSP2  Velocity Software-Test VSIVM4  ESAMON 3.778
1 of 3  User Percent Utilization                CLASS * USER

                              <-------Main Storage-------->
         UserID   <Processor> <Resident-> Lock <-WSSize-->
Time     /Class   Total  Virt Total  Actv  -ed Total  Actv
-------- -------- ----- ----- ----- ----- ----- ----- -----
00:11:00 ROBLNX1  32.39 32.38 15862 15862    11 15536 15536
         ROBLX2   32.12 32.11 66136 66136   259 78478 78478
         ROBLX1   32.02 32.01 38219 38219   176 37790 37790
```

[1] The 100 Hz timer has been replaced now by an on-demand timer, but many applications show a similar polling behavior.

We now give ROBLX2 a share setting of REL 200 because that is a more important service (nothing with virtual 2-way). We would expect it to get 50% and the others just 25% each.

```
Screen: ESAUSP2  Velocity Software-Test VSIVM4  ESAMON 3.778
1 of 3  User Percent Utilization                CLASS * USER

                                <-------Main Storage-------->
         UserID   <Processor> <Resident-> Lock <-WSSize-->
Time     /Class   Total  Virt Total  Actv  -ed Total  Actv
-------- -------- ----- ----- ----- ----- ----- ----- -----
00:14:00 ROBLX2   68.71 68.68 66211 66211   258 78478 78478
         ROBLX1   14.00 14.00 38245 38245   256 37790 37790
         ROBLNX1  13.99 13.99 15879 15879    11 15536 15536
```

The remaining 2 looping users now get far less (we would expect 2 x REL 100 to be the same as REL 200, but it’s more than double). Basically, all excess share goes to the virtual machine with the highest share willing to take it.

This would also happen when a customer defines his production servers as virtual 2-way but forgets to double the relative share. The effect is dramatic (worse than factor 2) when a test server goes into a loop or starts some heavy workload. Due to excess share, the test server gets as much CPU resources as it wants, while the production server is waiting. Confronted with this situation, some customers concluded z/VM is unable to ensure production response times when a test server misbehaves.

Be aware that LIMITSOFT of ABS 50% does not provide any relief. The CPU resources beyond the 50% that corresponds to its REL 200 are given already because of excess share. And without the fix for VM64721, even LIMITHARD does not work (and it is really breaking the idea of sharing resources when you must put up fences to ensure users get their minimum share).

Now for the experiment we reduce the relative share for all idle users to make it better match their actual CPU requirement. Instead, important service machines are given a small absolute share corresponding with their normal requirement to provide the service. The idea is that this improves the ability of the z/VM scheduler to predict the deadline. Obviously the default REL 10,000 and REL 1500 of some service virtual machines was also corrected.

After these adjustments, we look again at the CPU usage for our 3 looping users; the result is entirely as expected. This shows very nicely that the REL 200 is worth 2 times as much as REL 100.

```
Screen: ESAUSP2  Velocity Software-Test VSIVM4  ESAMON 3.778
1 of 3  User Percent Utilization                CLASS * USER

                                <-------Main Storage-------->
         UserID   <Processor> <Resident-> Lock <-WSSize-->
Time     /Class   Total  Virt Total  Actv  -ed Total  Actv
-------- -------- ----- ----- ----- ----- ----- ----- -----
00:20:00 ROBLX2   48.39 48.37 67141 67141   292 80047 80047
         ROBLNX1  24.19 24.19 16168 16168    11 15536 15536
         ROBLX1   24.19 24.18 39006 39006   241 37790 37790
```

And when we set ROBLNX1 to REL 300 it works again: 48%, 32% and 16% exactly like the REL 300, 200 and 100 we set for the users.

```
Screen: ESAUSP2  Velocity Software-Test VSIVM4  ESAMON 3.778
1 of 3  User Percent Utilization                CLASS * USER

                                <-------Main Storage-------->
         UserID   <Processor> <Resident-> Lock <-WSSize-->
Time     /Class   Total  Virt Total  Actv  -ed Total  Actv
-------- -------- ----- ----- ----- ----- ----- ----- -----
00:23:00 ROBLNX1  48.15 48.14 16170 16170    11 15536 15536
         ROBLX2   32.86 32.86 67190 67190   211 80047 80047
         ROBLX1   16.44 16.43 39016 39016   193 37790 37790
```

The experiment shows that with appropriate share settings, the z/VM scheduler is able to estimate the resource requirements. Consequently, virtual machines get access to the CPU resources according to their share settings. This is what customers expect to happen when they combine workloads with different business importance on a single z/VM image. To avoid confusing the scheduler, one should carefully assign relative share based on business requirements and be aware of the excess share distribution. When the installation wants a rather rude resource allocation (like “Development should only get resources when it is not needed for Production”) then it may still work effectively.

Until this is fixed in z/VM, relief could be found in an additional service machine that takes both z/VM and Linux performance metrics and adjust share settings in the proper way to guide the z/VM scheduler. ESALPS is obviously in good shape to do that.

An alternative approach could be to only change the number of virtual CPUs (assuming the Linux workload can exploit them) and keep the per-CPU relative share the same. While this increases overhead on z/VM and Linux a little, it may turn out to be more effective in allocating CPU resources.

Using cpuplugd

Popular configuration guidelines seem to encourage customers to define their Linux guests with many virtual CPUs. As we observed, in such a configuration Linux will spread the little work it has over all defined virtual CPUs.[2] So all CPUs get in queue each time, immediately after they consumed their time slice (or rather, gave up almost their entire time slice). This increases the amount of excess share in the system.

An interesting approach to address this problem is the cpuplugd tool. The idea is that it can be configured to vary off excessive virtual CPUs in Linux when there is little demand for CPU resources. While this was meant to improve single task throughput for Linux in LPAR, it could actually be useful to help the z/VM scheduler understand the proper CPU requirements of the virtual machine. When CPU requirement in Linux is low, the additional virtual CPUs are stopped and drop from queue and don’t suggest to be competing for resources anymore.

Starting with z/VM 5.4, the scheduler implements a “Share Redistribution” feature that breaks this model. Rather than showing the z/VM scheduler that the virtual machine has less need for CPU resources, this feature transfers the share of the stopped virtual CPUs to the remaining ones. This keeps the total in-queue relative share as high as before the virtual CPUs were stopped. And worse, it also grants that

[2] This is because Linux implements CPU affinity for the processes in an attempt to improve CPU cache hit ratio. Some of this may be influenced by other architectures than System z. It clearly is not an important aspect when the virtual machine shares the real CPU with hundreds of others.

remaining virtual CPU a much higher share of the CPU resources. This is especially painful when a single process in Linux loops and the load average[3] goes down to 1, that looping process is granted all excess share capacity (because varying off the other virtual CPUs raised its relative share setting above all others). Without this new z/VM 5.4 feature, cpuplugd would be a useful too to address the excess share problems. So we encourage z/VM Development to undo this change, or at least provide options for installations to disarm it.

This feature can also be abused by a Linux virtual machine get more CPU resources than the z/VM systems programmer meant to give it. We refer to “The Evil of Share Redistribution” since the full discussion is beyond the scope of this paper.

The research presented in this paper has been shared with z/VM experts of IBM.

Rob van der Heij

March 2010

[3] The prime control for cpuplugd is the “load average” or average number of runnable processes. This is not the best and most intuitive metric to use. See also our “Using cpuplugd to Allocate CPU Resources” paper.