Technical Investigation of Computational Resource Interdependencies

By Lars-Eric Windhab

Table of Contents

1. Introduction and Motivation

In clouds resources are deployed by virtual machines (VMs), i.e., if a cloud customer wants to run a job in a cloud, an according VM is started on a Physical Machine (PM) to execute the job. If the resources of a PM get overloaded, elaborated resource reallocation between the PM's VMs is necessary to ensure performance goals of VMs. To increase the efficiency of reallocations, it is necessary to account for interdependencies of resources and starvation limits of VMs. For example, a VM might need a certain minimum of RAM as otherwise its operating system crashes or becomes so slow that the other resources it utilizes are virtually of no use. Also, CPU and RAM may be required in a fixed ratio to start multiple threads. Contrary, it may also be the case that for a single threaded program additional RAM is of no use, but it greatly profits from more CPU cycles. If substitutions of resources are also considered, i.e., substituting RAM by disk space through paging or disk space or bandwidth by CPU cycles through compression, even more complex interdependencies can be deduced.

In economics such interdependencies are modelled by utility functions, i.e., a VM's utility function maps the resources it is allocated to a number that quantifies the performance it delivers with these resources for the workload assigned to it by its customer. While it is technically possible to quantify a VM's performance for the currently allocated resources it is challenging to determine its entire utility function, as this implies determining its performance with hypothetical resources without actually assigning them. However, utility functions are essential to allocate cloud resources efficiently, wherefore dependencies and starvation limits are to be investigated in the framework of this internship.

2. Problem to be solved

Utility functions of different entities that consume computational resources are to be investigated with a particular focus on entities relevant for cloud computing. Therefore, a comprehensive list of these entities and the types of workloads they may have to process is to be compiled. This list must highlight similarities, differences, and inclusions between the entities and workloads. From a technical analysis conclusions on the entities' utility functions and how they depend on the workloads are to be drawn and formalized. The technical analysis must also support the conclusions by practical measurements/experiments.

3. Discussion of design choices

This section serves as justification for our design choices

3.1 Definition of computational resources

In order to deduce utility functions to enable fair resource allocation for different cloud workload scenarios it is necessary to determine relevant computational resources that are required for execution first.
These resources can be divided into a physical and a virtual category. Since virtual resources such as file handles ultimately are limited by their physical counterpart we concentrate our efforts on the latter.
The most relevant physical computational resources, in terms of performance, are the central processing unit, the memory and the network bandwidth.
Their consumption levels can but do not have to correlate. Correlation is determined by the consumption structure of an executed process and may change dynamically.
The usage of the CPU and RAM by a process for instance, might change in a fixed ratio until it reaches a point where a new thread increases the CPU usage but does not require additional RAM or vice versa [**1**]. Moreover it is possible that resources can be used to substitute each other [**2**].

3.2 Measuring the resource consumption of a cloud workload

3.2.1 The test system

Host machine:
OS: Linux Ubuntu 14.04 Trusty Thar
CPU: Intel i7-3770 @ 3.4 GHz
CPU Cores (Physical / Logical): 4/8
Memory: 8 GB

Virtual machine:
OS: Linux Ubuntu 14.04 Trusty Thar
CPU Model: Sandy Bridge
CPU Cores (Virtual): 1,2,4,8
Memory: 1 - 8 GB

3.2.2. How to measure computational resource consumption

The resource consumption of a computer (real or virtual) can be measured at the process level, where its total consumption equals the aggregated consumption of each process it executes.
During this project we use a Linux based host system and VM (see 3.2.1).
There are a number of available tools that can be used to measure resource consumption but we found none that covers all aspects that we want to monitor. We therefore calculate the resource consumption ourselves by repeatedly parsing and interpreting the data of the /proc/ kernel interface [**3**] with the help of a specifically developed monitoring tool. The monitoring tool is written in java and can be executed via the command line interface. It is possible to pass parameters to choose whether only one process or all of them should be monitored, the length of the observation and the relevant network adapters. Additionally it will output the data to a text file to prepare it for plotting.

Resource	**Data**	**Description**	**Unit**
CPU	CPU activity	CPU is activity in a 0.5 second interval	%
Memory	Allocated memory	Change of allocated memory in a 0.5 second interval	kB/s
Memory	Allocated memory	Allocated memory since the start of measurement	kB
Memory	Allocated memory	Total memory that is allocated in the system	kB
Memory	Allocated memory	Total memory is allocated in the system	%
Bandwidth	Download	Received network data in a 0.5 second interval	kB/s
Bandwidth	Upload	Sent network data in a 0.5 second interval	kB/s
Bandwidth	Download	Total amount of received network data	kB
Bandwidth	Upload	Total amount of sent network data	kB
Disk	Disk activity	Disk activity in a 0.5 second interval	%
Disk	Data written	Amount of data written on disk in a 0.5 second interval	kB/s
Disk	Data read	Amount of data read from disk in a 0.5 second interval	kB/s
Disk	Data written	Amount of data written on disk since start	kB
Disk	Data read	Amount of data read from disk since start	kB

Table 1 - Overview of the monitor tool output

```
lareida@ubuntu:~$ java -jar monitor.jar
monitor -i (iterations) -a (network adapters)
Press Control-C to stop.
Interval: 500.0 ms Iterations: 100 Network Adapters to scan: 4
CPU%: 5.0 %; MEM: 4544.0 kB/s; MEM Used: 6058836.0 kB; MEM%: 74.0 %; Down: 0.0 kB/s; Up: 0.0 kB/s; Disk: 0.0 % Disk Total: 0.0 kB
CPU%: 4.0 %; MEM: 3488.0 kB/s; MEM Used: 6061356.0 kB; MEM%: 75.0 %; Down: 0.0 kB/s; Up: 0.0 kB/s; Disk: 0.0 % Disk Total: 0.0 kB
CPU%: 6.0 %; MEM: 336.0 kB/s; MEM Used: 6063100.0 kB; MEM%: 75.0 %; Down: 0.0 kB/s; Up: 0.0 kB/s; Disk: 5.0 % Disk Total: 64.0 kB
CPU%: 2.0 %; MEM: 0.0 kB/s; MEM Used: 6063268.0 kB; MEM%: 75.0 %; Down: 0.0 kB/s; Up: 0.0 kB/s; Disk: 0.0 % Disk Total: 64.0 kB
CPU%: 4.0 %; MEM: -128.0 kB/s; MEM Used: 6063268.0 kB; MEM%: 75.0 %; Down: 0.0 kB/s; Up: 0.0 kB/s; Disk: 0.0 % Disk Total: 64.0 kB
CPU%: 6.0 %; MEM: 32.0 kB/s; MEM Used: 6065244.0 kB; MEM%: 75.0 %; Down: 0.0 kB/s; Up: 0.0 kB/s; Disk: 0.0 % Disk Total: 64.0 kB
CPU%: 6.0 %; MEM: 512.0 kB/s; MEM Used: 6065260.0 kB; MEM%: 75.0 %; Down: 0.0 kB/s; Up: 0.0 kB/s; Disk: 0.0 % Disk Total: 64.0 kB
CPU%: 5.0 %; MEM: 0.0 kB/s; MEM Used: 6065516.0 kB; MEM%: 75.0 %; Down: 0.0 kB/s; Up: 0.0 kB/s; Disk: 0.0 % Disk Total: 64.0 kB
```

Figure 1 - Example output of the monitor tool

3.2.3 Process vs. thread

A process itself could be broken down further into threads, since a thread can be described as a subprocess of a process and is the smallest sequence of programmed instructions that can be managed dependently by an operating system scheduler [**4**]. However the expense of analyzing separate threads bears no proportion to the information gained. This is because threads generally share the process memory (apart from the thread local storage which is nearly exclusively used to hold a reference to an object in the shared memory) and their aggregated resource usage is visible via the parent process [**5**][**6**]. We therefore conclude that the process level approach is sufficient for our experiments.

3.2.4 Where to measure

A workload can be described as all individual units of work that constitute a discrete application [**7**]. A typical cloud workload thus consists of the middleware such as the orchestration layer and hypervisor (Xen, KVM, VMware ESX) and the virtual machine itself [**8**].

In our case the middleware consists of the KVM.
However, before a qualitative analysis of different cloud workloads can be begun the influence of KVM needs to be analyzed in order to determine if it is feasible to include it in our measurement.

To do so, it is possible to monitor the consumption of the virtual machine process in the host OS (1), because it is the aggregation of all relevant workload processes performed inside the VM including the middleware, or to aggregate the consumption of all processes in the VM itself excluding the middleware (2).

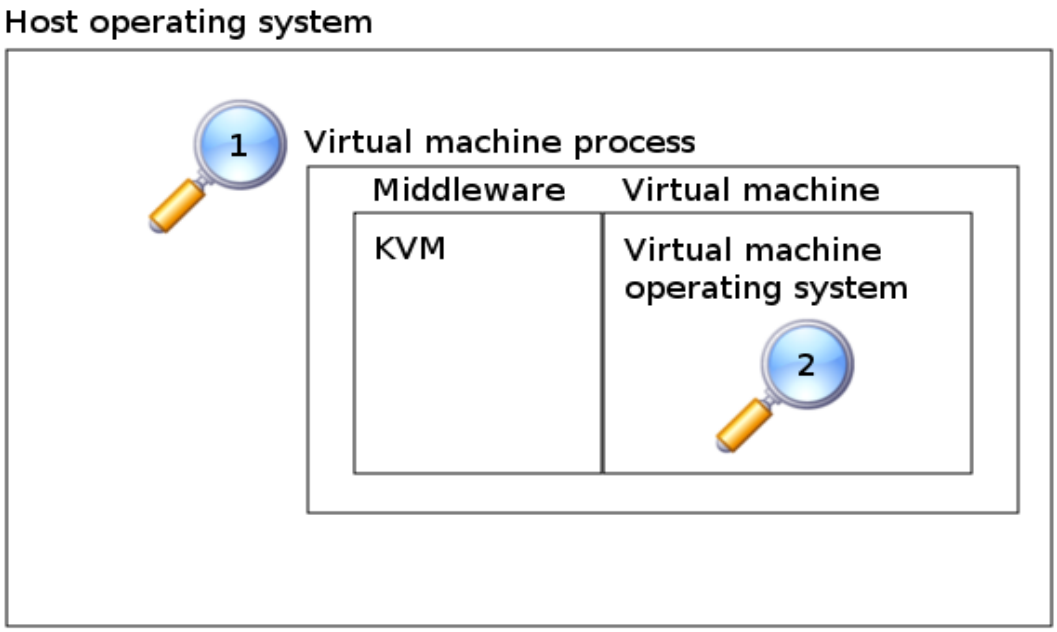

Figure 2 - Abstraction of the components required to run a virtual machine

A series of tests were performed, while we monitored the **VM Process**' resource consumption in the host OS and inside the VM itself (**VMOS**).

CPU:

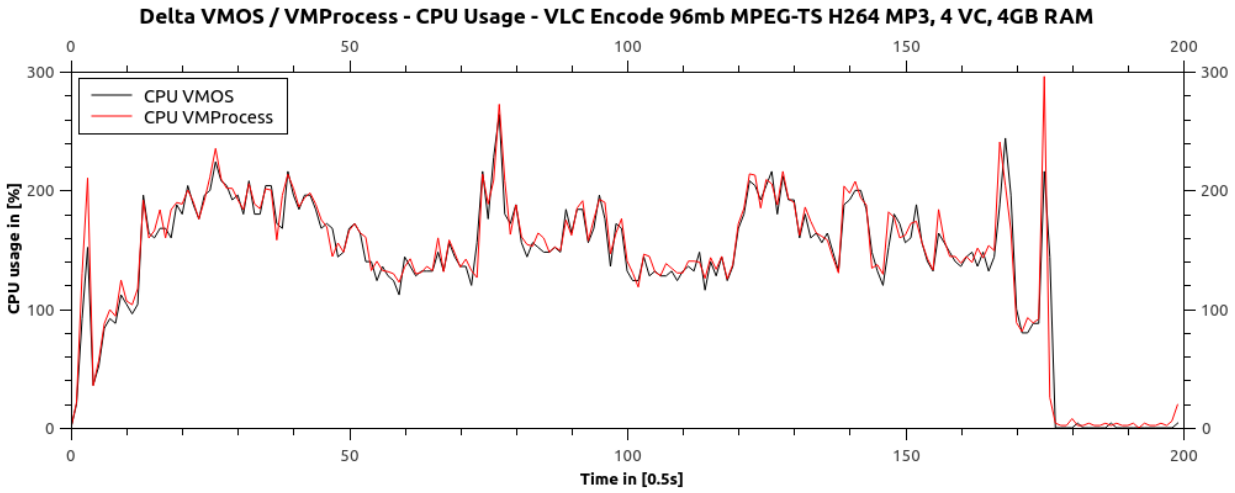

Figure 3 - Influence of the middleware on CPU consumption

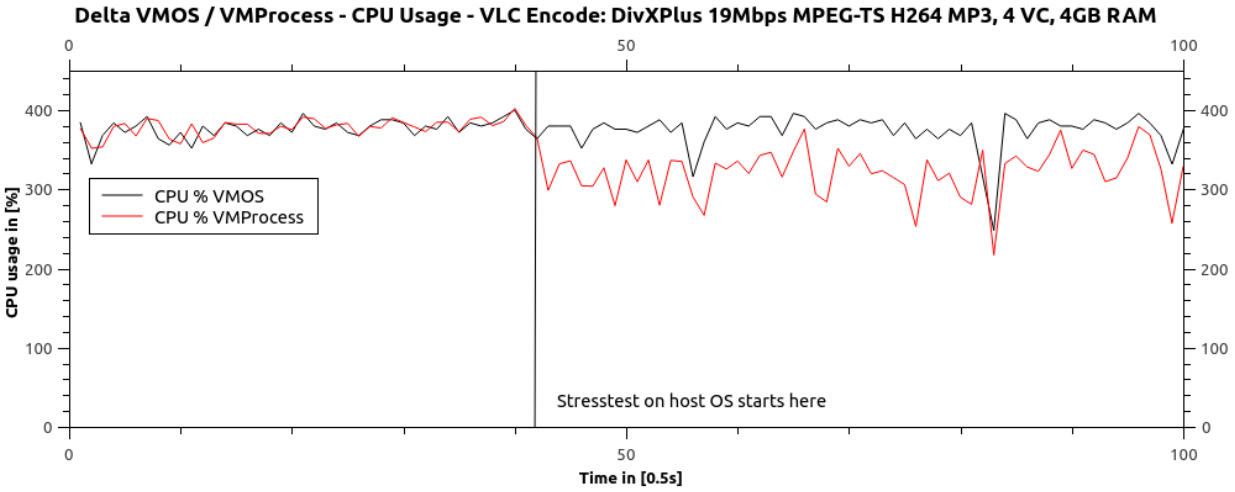

Figure 4 - Erroneous CPU values in the VM when CPU is being limited in the host OS

Memory:

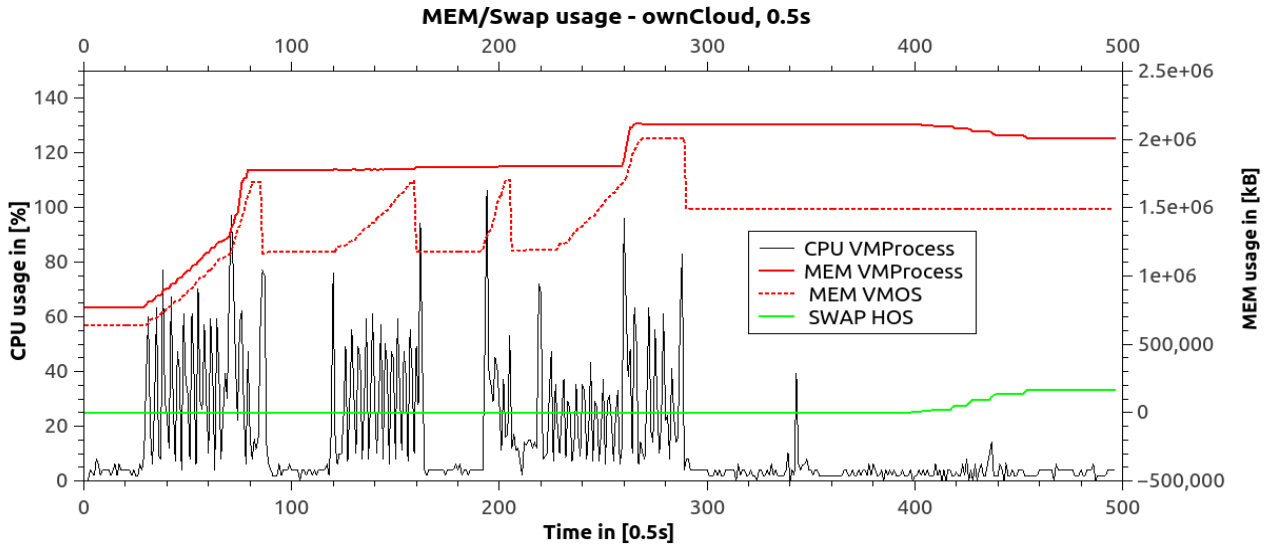

Figure 5 - Memory overload on host OS causes no memory deallocation for the VM Process

Bandwidth:

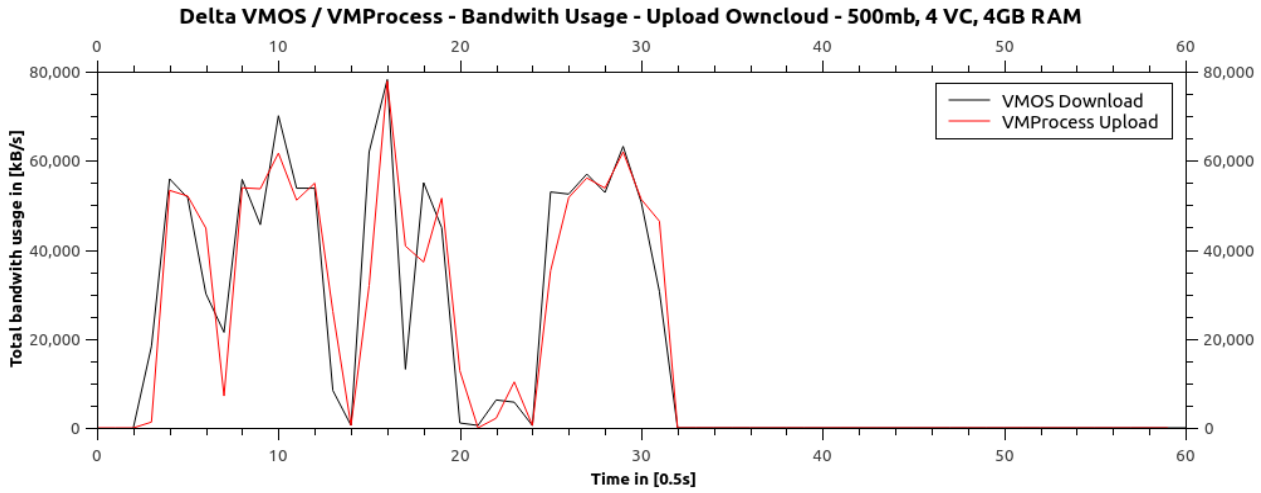

Figure 6 - KVM influence on bandwidth usage

Disk:

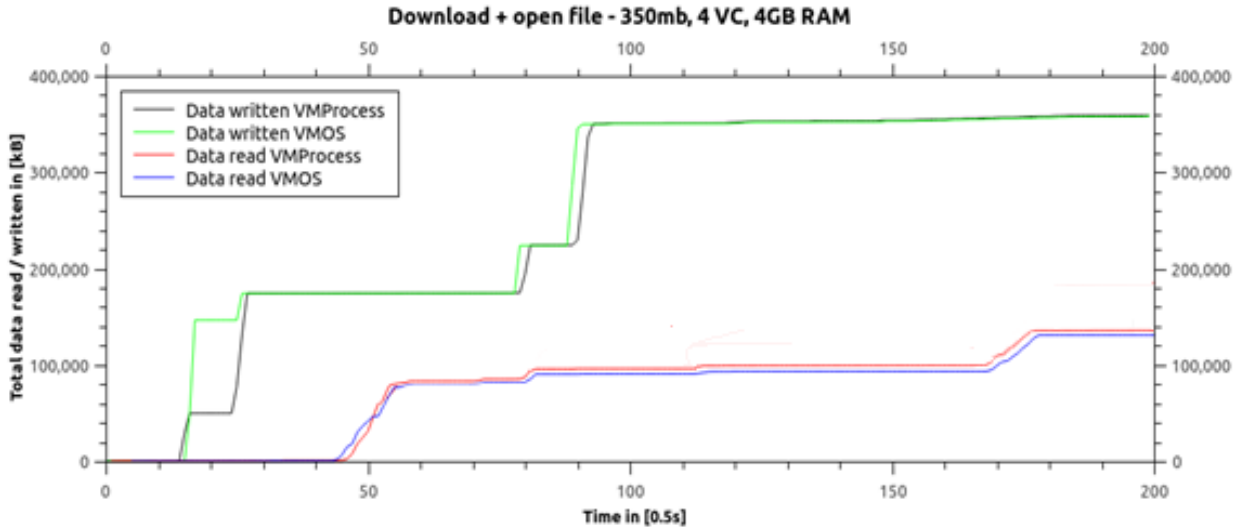

Figure 7 - KVM influence on disk stats

An almost perfect correlation between the VMOS and VM Process is apparent in the CPU usage for the encoding workload [**Figure 3**]. If we however use the CPU in the host OS to capacity, by running a stress-test on 8 cores, we can observe that the VMOS reported CPU usage starts to be higher compared to the actual host OS values [**Figure 4**].
Furthermore, a significant difference in RAM allocation is visible. The VMOS frees unused RAM while the VM Process will not do so . Even if we overload the host OS' memory (simulating a resource shortage), the allocated but unused memory of the VM Process will not be released. Instead memory in the host OS is swapped [**Figure 5**].
In terms of bandwidth usage no differences were observed [**Figure 6**] the same holds true for the disk in and output where only a small delay when writing the data is observed. [**Figure 7**].

While we assume that the RAM disallocation problem is caused by the middleware (KVM), we believe that the differences in reported CPU usage are caused by problems within the proc/stat kernel interface which is not working correctly, in terms of CPU usage, within a virtual machine [**9**].

These findings have implications on our design.

To ensure the best accuracy of results, we decided to not include the middleware in our measurements and solely focus on the vm executing the workload.
This has two reasons. First, the middleware varies between cloud hosts, so analyzing a particular hypervisor bears no value when looking for general patterns and second, in the special case of KVM it is not feasible to include it when detailed RAM statistics are required because of the distortion it causes.
On the other hand, CPU values from within a VM can be erroneous but are not altered by the KVM, thus must be measured in the host OS, to get the most accurate results.
Bandwidth usage can potentially be measured in both environments.

This justifies our setup where CPU consumption is measured in the host OS including the middleware while bandwidth, disk I/O and RAM statistics are gathered inside the VM.

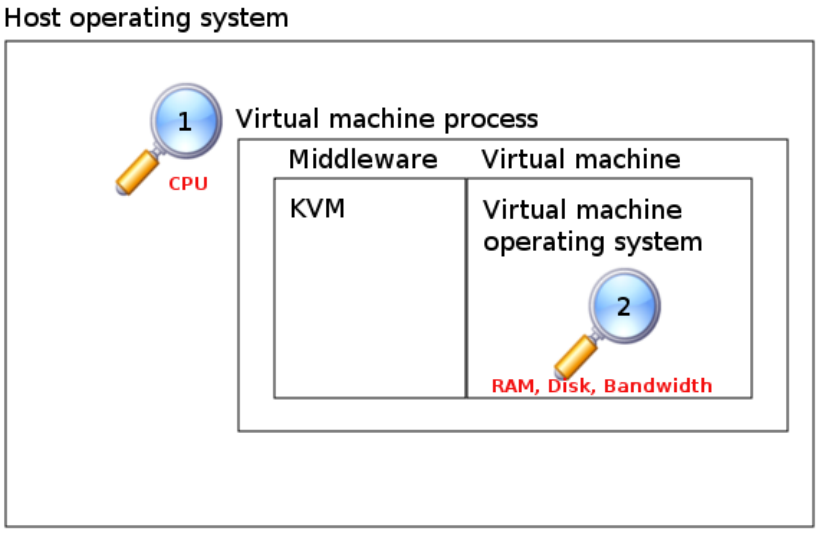

Figure 8 - The final measuring setup

In principle it would be possible to run the workload directly in the host OS and measure consumption there if we want to disregard the middleware. However, to isolate the workload and test different resource configurations with the possibility of altering them dynamically, it is best run in a VM.

3.2.5. Simulating a cloud workload

The testing framework consists of a main machine (including host OS and VMs) and a remote machine.
Workloads for the VM are generated in the remote machine to minimize any influence in the host OS.
Since consistency in generating workloads is important to produce comparable and reproducible results we used JMeter [**10**] to standardize and execute them.

3.2.6 Limiting resources

To understand how a workload utilizes different resources we must investigate it while varying the resource parameters. We can change resource allocations by creating different virtual machines, or we can limit them dynamically by running stress tests or a bandwidth limiter [**11**, **12**] in the host OS.
An interesting observation we made is that statically hosting a VM with less available resources equals limiting the available resources in the host OS.

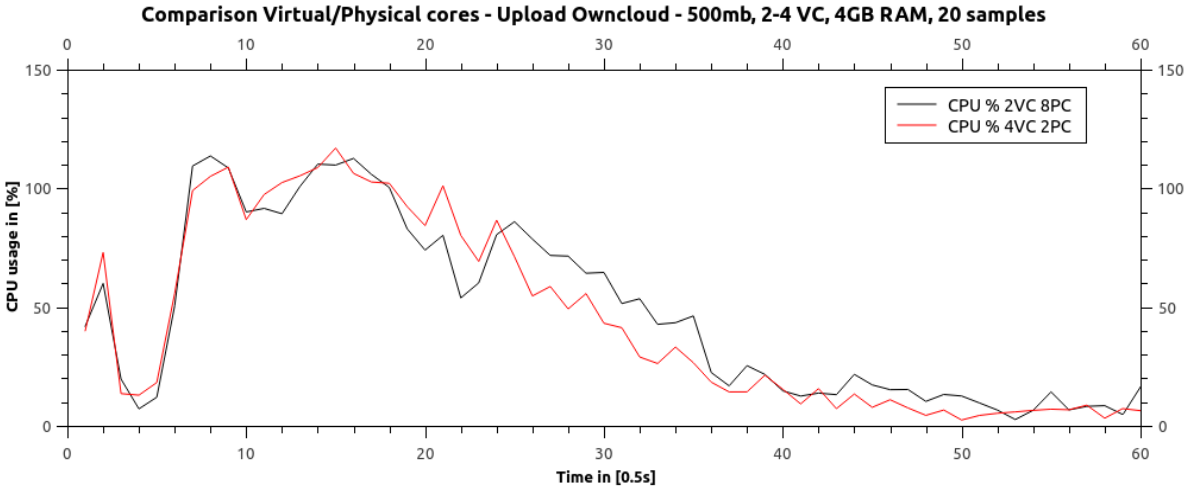

Figure 9 - Comparison of a 2VC VM setup and limiting physical CPU to 2 cores

3.2.7 Sample size

We collected at least 8 samples of each workload scenario and then used the average for our study.

3.3 Which workloads are investigated?

In order to create utility functions representative workloads of common cloud applications need to be investigated. The following table lists an overview of the examined workloads.

Application Name	Workload 1	Workload 2	Workload 3
ownCloud	Upload - 500mb file	Download - 500mb file	Delete - 500mb file
VLC Player	Stream - 96mb file (DivX 720p)	Encode - 96mb file (DivX 720p) to MP4	
MySQL	Update table - 5000 queries		

Table 1 - Workloads

4. Solved and open issues

4.1 Results

CPU limitation

Workload	CPU	RAM	Disk - read	Disk - written	Bandwidth - up	Bandwidth - down	Execution time
ownCloud *upload*	decreased demand	increased demand	-	-	-	-	-
ownCloud *download*	increased demand	-	-	-	-	-	increased
ownCloud *delete*	-	-	-	-	-	-	-
VLC *stream*	-	-	-	-	-	-	-
VLC *encode*	increased demand	-	-	-	-	-	increased
MySQL *update table*	-	increased demand	-	-	-	-	-

Table 2 - Workload behaviour when CPU is limited

RAM limitation

Workload	CPU	RAM	Disk - read	Disk - written	Bandwidth - up	Bandwidth - down	Execution time
ownCloud *upload*	increased demand	-	-	-	-	-	increased
ownCloud *download*	increased demand	-	-	-	-	-	increased
ownCloud *delete*	increased demand	-	-	-	-	-	increased
VLC *stream*	-	-	-	-	-	-	-
VLC *encode*	increased demand	-	-	-	-	-	increased
MySQL *update table*	increased demand	-	-	-	-	-	increased

Table 3 - Workload behaviour when RAM is limited

Disk limitation

Workload	CPU	RAM	Disk - read	Disk - written	Bandwidth - up	Bandwidth - down	Execution time
ownCloud *upload*	increased demand	-	-	-	-	-	increased
ownCloud *download*	decreased demand	-	-	-	-	-	increased
ownCloud *delete*	decreased demand	-	-	-	-	-	increased
VLC *stream*	-	-	-	-	-	-	-
VLC *encode*	-	-	-	-	-	-	-
MySQL *update table*	increased demand	increased demand	-	-	-	-	increased

Table 4 - Workload behaviour when Disk is limited

Bandwidth limitation

Workload	CPU	RAM	Disk - read	Disk - written	Bandwidth - up	Bandwidth - down	Execution time
ownCloud *upload*	increased demand	-	-	-	-	-	increased
ownCloud *download*	increased demand	-	-	-	-	-	increased
ownCloud *delete*	-	-	-	-	-	-	-
VLC *stream*	error	error	error	error	error	error	error
VLC *encode*	-	-	-	-	-	-	-
MySQL *update table*	increased demand	-	-	-	-	-	increased

Table 5 - Workload behaviour when Bandwidth is limited

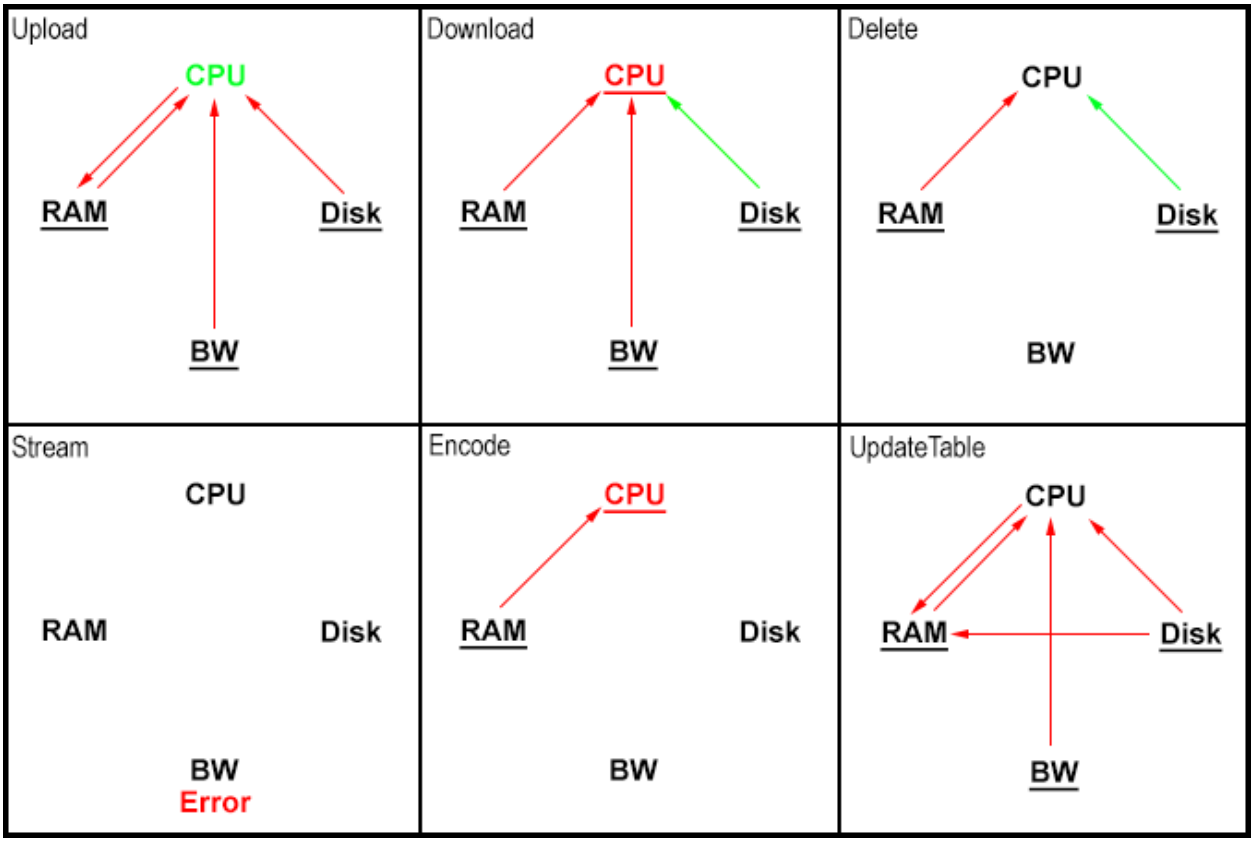

Figure 10 - Visual comparison of the results

4.2 Conclusion

In terms of CPU limitation we observe two effects if a bottleneck is created. The first being an increase in RAM usage while CPU usage stays constant or decreases. We assume that the lacking CPU resource is substituted with RAM. The second effect, when no substitution takes place, is that the workload will be executed slower with an overall increased need of CPU cycles.
Some workloads were not affected by the CPU limitation at all, we attribute this to the fact that the limitation was not effective enough to cause a lack of CPU resources for the given workloads.
When we look at limiting RAM a very homogenous pattern is apparent. For all but one workload it causes an increase in CPU demand while extending the needed amount of time to execute the workload. In the case of streaming a video, RAM limitation does not alter performance or consumption. This might be caused by us not being able to create a significant RAM shortage since the streaming workload is, by nature, bound to execute during a fixed amount of time and thus slow allocation of RAM, as it happens when we overload the host OS RAM and force it to swap, does not impact the workload since its RAM usage increases slowly.
As for disk I/O limitation, different patterns with overlapping features emerged. While we still encounter workloads that are not affected for previously stated reasons, those that were have one thing in common namely a prolonged execution time. However, we observed that this prolonged execution time with slower I/O can either cause an overall de- or increase of CPU cycles. The case of the MySQL workload shows us that RAM usage can be affected by limited I/O too.
Effects caused by limiting the Disk I/O seem to be less distinct.
In the case of limiting bandwidth, the results were as we expected them to be. For the tasks that require bandwidth to execute an increase in execution time as well as CPU cycles used was observed. In the case of the streaming workload the limitation caused it to not execute properly at all. This again can be attributed to the fixed amount of time required to execute the workload properly which is not possible due to the limitation.

4.3 Open issues

The findings suggest that leontief preferences to model resource consumption in VMs are a simplification at best. However, in order to formalize the results more measurements with different limitation values need to be conducted

5. Critical consideration of the task, the work and result

The initial idea behind the project was our assumption, that different workloads do need different resources in different compositions in order to execute. We have shown that this is indeed the case. Furthermore we were able to highlight similarities, differences, and inclusions between the entities and workloads. However, the task itself was very broadly defined and therefore a lot of assumptions had to be made. For instance which operating system and VM ware fits our purposes best and which workloads are most representative and how we define a workload.
The investigation of resource consumption itself can be split up into two subtasks.
The first is finding an optimal setup, which was characterized by a trial and error approach. In order to find out how to properly measure most accurately our design had to be revised multiple times.
Once a stable design was found the second subtask which is the actual measuring could be conducted without further problems.
Allthough we were able to identify consumption patterns that we can explain, we encountered effects that are beyond our understanding. These results have to be analyzed in detail and could be subject to further work. Furthermore more workloads could be investigated as well as different setups regarding the middleware.

6. References

[1] http://superuser.com/questions/78362/what-is-the-relationship-between-cpu-usage-and-ram (13.09.2014)
[2] No Justified Complaints: On Fair Sharing of Multiple Resources
[3] http://www.linux-praxis.de/lpic1/lpi101/proc.html (13.09.2014)
[4] http://en.wikipedia.org/wiki/Thread_(computing) (13.09.2014)
[5] http://www.programmerinterview.com/index.php/operating-systems/thread-vs-process/ (13.09.2014)
[6] http://en.wikipedia.org/wiki/Thread-local_storage (13.09.2014)
[7] http://www.devx.com/blog/understanding-cloud-workloads.html (13.09.2014)
[8] http://www.dummies.com/how-to/content/types-of-workloads-in-a-hybrid-cloud-environment.html (13.09.2014)
[9] http://linuxvm.com/topisbad.html (13.09.2014)
[10] http://jmeter.apache.org/ (13.09.2014)
[11] http://people.seas.harvard.edu/~apw/stress/ (13.09.2014)
[12] http://lartc.org/wondershaper/ (13.09.2014)
[13] http://aws.amazon.com/de/ec2/instance-types/ (13.09.2014)

7. Acknowledgements

I would like to thank Patrick Poullie and Thomas Bocek for their input and guidance during this project.

8. Appendix

A CD with the montioring tool and its source code.