A Novel Framework for Simulating Computing Infrastructure and Network Data Flows Targeted on Cloud Computing

Peter Krauß
Karlsruhe Institute of Technology
Steinbuch Centre for Computing
76128 Karlsruhe
Email: peter.krauss@kit.edu

Tobias Kurze
Karlsruhe Institute of Technology
Library
76131 Karlsruhe
Email: kurze@kit.edu

Achim Streit
Karlsruhe Institute of Technology
Steinbuch Centre for Computing
76128 Karlsruhe
Email: achim.streit@kit.edu

Bernhard Neumair
Karlsruhe Institute of Technology
Steinbuch Centre for Computing
76128 Karlsruhe
Email: bernhard.neumair@kit.edu

***Abstract*— Understanding how computing infrastructure decisions influence overall performance and cost can be very difficult. Simulation techniques are an important tool to help analyse and evaluate different infrastructure configurations and deployment scenarios. As diversity of cloud computing resources is growing, the space of possible infrastructure configurations becomes larger and finding the best solution becomes harder. In our previous paper, we presented a framework to benchmark cloud resources to obtain objective and comparable results. Based on those results, our simulation framework allows to model applications and to estimate their performance. Compared to other infrastructure or cloud simulation tools our framework excels when it comes to the simulation of network data flows.**

Keywords–Broad band networks, quality of service, infrastructure, cloud, simulation, framework

I. INTRODUCTION

Due to the rise of cloud computing during the last few years - for a widely accepted definition of the term see [1][2] - and its further growing adoption in commercial, industrial and research endeavors [3][4], there are more and more different infrastructure offerings and possible applications. Different types of applications have different requirements on the underlying (virtual) hardware and are structured differently. Finding deployment policies that deliver good performance and are cost-efficient under varying conditions is challenging because of multiple reasons.

First, the number of possible deployments for a given application respectively for workload executing systems, grows rapidly with the number of components of an application as well as with the number of available resources to choose from. Secondly, clouds do not always deliver the same performance and are subject to varying demand. Further, as cloud resources are delivered via public broadband networks, it is hard or even impossible to reproduce identical test conditions. As it is neither practical to try out each and every possible deployment nor feasible to reproduce an identical test environment, simulating the respective scenarios is the better approach. Therefore, we developed an infrastructure simulation framework that focuses on cloud environments, but in principle it can be used for any kind of infrastructure. The framework allows the simulation of large-scale applications that may be deployed across different cloud providers.

Our framework provides informations about the complete behavior of a configured workload running on a user defined infrastructure. During the simulation, a user can access all the status information he might need, this includes load on disk, CPU, memory as well as network interfaces. This allows a user to attach e.g., cost models or simulate bad or malfunctioning hardware. Our approach is unique in the way it models networks and is very lightweight compared to other tools.

The paper is structured as follows:
A short overview of cloud respectively infrastructure simulation tools is given in Section II. Next, we present our infrastructure model and introduce resource providers and our network simulation model. Then, we give a description of how workloads are specified, followed by the section dedicated to the implementation of the simulator. Next, in addition to the theoretical model, we describe how calibration of the simulator is done and show the validity of the framework. Lastly a short summary including a discussion of the validation's results is given.

II. RELATED WORK

In the field of cloud simulation, a lot of research with varying focus has already been done. Lim et al. developed MDCSim [5], a simulation platform for multi-tier data centers that allows performance and power consumption analysis. The simulator is event based and designed as a three-layer architecture – user layer, kernel layer, communication layer. It is able to capture details such as kernel level scheduling

 ISBN: 978-1-61208-460-2

strategies. For communication, the simulator supports IBA and Ethernet over TCP/IP. By changing the timing parameters other protocols can be incorporated. For kernel requests, the simulator considers multiple CPUs as a single resource and is modeled according to the Linux scheduler 2.6.9.

Another well known toolkit for modeling and simulating cloud environments is CloudSim [6][7]. It is event-based and allows to model data centers, virtual machines and resource provisioning policies. Furthermore it is possible to model federation of clouds resp. inter-networked clouds. Allocation of virtual machines to hosts is done based on a First-Come-First-Serve basis. CloudSim implements time-shared as well as space-shared policies at host and at VM level to assign resources. Besides aforementioned aspects, CloudSim models the cloud market based on a multi-layered design, whereby the first layer represents costs per unit related to the IaaS model and the second layer comprises costs related to the SaaS model. Network behavior is modeled using a latency matrix and a message will be forwarded by the event management engine after a delay specified by the according entry in the latency matrix.

NetworkCloudSim [8] has been developed by Garg and Buyya and is an extension for CloudSim providing a network and generalized application model, allowing a more precise evaluation of scheduling and provisioning policies. NetworkCloudSim adds three main entities: Switch, NetworkDatacenter and NetworkDatacenterBroker. Also new classes to model networks have been added to the original classes of CloudSim. This allows a better modelling of applications that rely on specific means of communication such as MPI for example.

DartCSim+ [9] is another enhancement of CloudSim. The authors claim to have overcome limitations of CloudSim in regard to three aspects: – simultaneous support of power and network model is not possible – simulation of network components is not power-aware – migration does not take into account network overheads To overcame aforementioned limitations, the authors developed a range of entities as extension for CloudSim.

Wickremasinghe et al. propose another CloudSim related tool named CloudAnalyst [10], that helps studying the behaviour of large-scale cloud applications. CloudAnalyst takes into account the geographic dìstribution of an application and of its users. CloudAnalyst provides a graphical user interface that allows users to model their experimental setups.

Nunez et al. developed another cloud simulator called iCanCloud [11][12] with focus on lage experiment setups. iCanCloud features a global hypervisor that can integrate any brokering policy and is configurable through a graphical user interface. It also provides a POSIX-based API and supports configurations for different storage systems such as remote file systems like NFS or parallel files systems and RAID configurations.

In our own previous work [13] we created a performance measuring tool. The publication contains a description of the used architecture to monitor a large amount of cloud resources over time. We were able to show that the perfromance of virtualized infrastructure resources does vary over time, even though it should be constant, as well as the performance of network interconnects between virtual machines. This aspect is not covered by common simulation frameworks. The framework presented in this paper is capable of taking the factors shown in [13] into account. We use the measured performance values from [13] to calibrate our simulation framework presented in this publication.

III. Infrastructure Model

Our model describes an infrastructure cloud environment as a hierarchy of entities. The entities on the lowest level provide resources, which are in our model *calculation power* in units called *cpu cycles*, *ephemeral memory* in units called *mbytes*, *persistent storage* space in multiples of mbytes and finally *network traffic* which is described by transfers between network enabling entities characterized by a bandwidth in units of *mbytes per time step*. The entities that provide those resources are called *vCPU*, *vMemory*, *vStorage* and *vNet* which creates instances of *vTransfer* objects.

On the next level, the model introduces another entity called *vMachine*. This entity can be associated with any combination and any amount of entities of the level below and is a loose equvivalent to a virtual machine in a real cloud. Further a *vMachine* can be associated with so called *vWorkload* objects. Those objects contain a description of a workload and are described in the course of this paper (see Section IV).

The next layer contains *vDatacenter* objects that in turn contain *vMachine* objects. A *vDatacenter* in addition is associated with two coordinates that express a location in a 2D-space and can be used to position different datacenters relative to each other. The distance between datacenters is factored in when data transfers happen, since previous works showed that this can affect transfers [13].

Beside those layers, the model is based on a global environment object containing all entities related to the simulated infrastructure. This environment provides a global, steadily increasing counter to represent the simulation time in logical seconds. With each step the timer advances by one and the simulation progresses. The environment makes sure that all objects associated with it are notified at least once per time step about the new simulation time so they will update and possibly advance their state.

In the following subsections, major parts of this model will be explained in deeper detail.

A. Resource Providers

As mentioned before the model is based on four resources that are provided by four resource providers. These entities are always associated with a maximum capacity and a provisioning speed. The first describes how many units of a certain resource are provided whereas the latter expresses at what speed this resource can be delivered. In addition, each resource can either be ephemeral or static. Ephemeral resources reset their capacity at the beginning of each new time step while static resources do not. In our model the following four resource providers are defined:

vCPU As mentioned, this object provides *calculation power* in units of *cpu cycles*. The resource is ephemeral and resets to its predefined maximum capacity with every

simulation step and the resource can be consumed at instant speed.

vMemory To represent a machine's internal memory, we introduced a *vMemory* resource provider. In contrast to physical RAM, this resource does reset its capacity with every time step due to how workloads are designed in our model: A *vWorkload* does not allocate and free memory but it provides an absolute value representing how much memory is required at a certain point in time. All operations on a *vMemory* object are performed instantly.

vStorage A *vStorage* object represents a comparably slow and limited storage device for example a hard disk or solid state drive. The provider's operational speed is capped at a predefined rate (throughput) which in addition is shared among all entities currently using the resource.

vNet This object represents a network device which can transfer a given amount of data per time step. The object's bandwidth is shared among all currently active transfers and each transfer's speed further depends on the receiver's bandwidth. Since proper network simulation is a major part of our framework this will be discussed in detail in a following section (see Section III-B). We assume a physical network card to operate in duplex mode, sending and receiving speeds are not correlated in any way and are thus modeled as two independent vNet instances that may be configured with different bandwidths.

vTransfer vTransfer objects contain all information that describe the state of ongoing data transfer, such as current transmission speed and associated vNet instances (origin/destination). In contrast to the previously presented objects, vCPU, vMemory, vStorage and vNet, a vTransfer does not have a representation in real hardware and is a completly virtual construct.

Since access to the 'fast' resources *calculation power* and *ephemeral memory* is considered to be performed instantly, no sophisticated scheduling is needed: In case those resources are requested by one or more workloads, the resource provider simply evenly divides the available resources among the requesters. In our current implementation, the scheduler follows a round-robin manner to divide the resources. In case a request cannot be fulfilled, an appropriate notification is sent to the *vWorkload* for it to react according to its configuration. For example, a common workload will request as many CPU cycles as possible and a certain amount of memory. As long as the first request does not lead to a critical error, a typical workload can still execute even if there's only a few cycles assigned to it. On the other hand, a workload will most likely crash if a memory requirement cannot be met.

In contrast, access to the 'slow' resources, storage and network, has to be scheduled. We model two operations on storage devices: `write` and `read`. Both operations are performed equally but in the first case, the capacity of the resource provider is decreased by the amount of transferred data whereas in the latter case, the capacity remains constant. When an operation on such a storage resource is initiated, the resource provider will register the operation and evenly divide the available bandwidth among the contenders and process them in parallel. An operation is considered as failed, if the requested amount of data cannot be written or read.

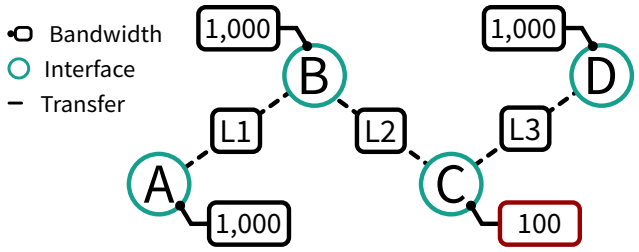

Figure 1. A simple example of a network graph.

B. Network Simulation

One of the major concerns of the simulator described in this paper is the network simulation model. As indicated above a proper calculation of per-transfer speeds can be difficult due to the dependency on the sender's as well as the receiver's bandwidth. The complexity increases with the number of transfers, which, in addition, might or might not share the same destination. A simple example is depicted in Figure 1. The example shows four vNet objects named A, B, C and D associated with the bandwidths $bw_A = 1{,}000$, $bw_B = 1{,}000$, $bw_C = 100$ and $bw_D = 1{,}000$ and three transfers between those ($L_1 = B \rightarrow A$, $L_2 = B \rightarrow C$ and $L_3 = C \rightarrow D$). The data transfers are represented as instances of *vTransfer*. It is obvious that the bandwidth of B has to be shared among two transfers L_1 and L_2 while being capped by the available bandwidth of C after taking L_3 into account. The arising challenge is to balance the transfer speeds in a realistic manner in real-time while considering that the network configuration might significantly change between time steps due to added or removed nodes or transfers. Due to those constraints known algorithms like those based on the *max-flow/min-cut*-theorem [14] are not suitable for our environment. Instead we developed an agent-based algorithm that determines the available bandwidth per active transfer at the current simulation time solely based on the information provided by the receivers of the transfered data.

The algorithm we propose uses an iterative approach that converges to a solution where an optimal data flow between all vNet objects is achieved. This algorithm is further based on a graph whose vertexes $M = \{M_1, \ldots, M_n\}$ are representing data sending or data receiving network interfaces (vNet objects) and whose set of edges $L = \{L_1, \ldots, L_n\}$ are representing network transfers (vTransfer objects). Each vertex M_i is associated with a limited transfer rate $bw(M_i) = bw_i > 0$ expressing the bandwidth of the vNet. The set of vTransfers N_i contains all vTransfers originating from M_i and is a subset of L.

To calculate the current transfer speed s_t at time t of a vTransfer object $L_k = L_{i \rightarrow j} = L_{M_i \rightarrow M_j}$ with $L_{i \rightarrow j} \in N_i$ originating from $o(L_k) = M_i$ and targeted to $d(L_k) = M_j$, the algorithm works in an iterative manner realistically saturating the interfaces.

In a first step ($t = 0$), for each vNet instance all available bandwidth is divided proportionally among the active vTransfers based on the theoretically possible transfer speeds to the corresponding remote partners which is assumed to be defined by the speed of the slowest involved vNet. This can

TABLE I. A SAMPLE CALCULATION FOR THE NETWORK GRAPH SHOWN IN FIGURE 1.

		L_1	L_2	L_3
1st Iteration	A	1000	–	–
	B	909	91	–
	C	–	50	50
	D	–	–	1000
	min()	909	50	50
2nd Iteration	A	1000	–	–
	B	948	52	–
	C	–	50	50
	D	–	–	1000
	min()	948	50	50
3rd Iteration	A	1000	–	–
	B	950	50	–
	C	–	50	50
	D	–	–	1000
	min()	950	50	50

be expressed in the following equation:

$$s'_0(L_{i\to j}) = s'_{t=0}(L_{i\to j}) = \frac{bw_j}{\sum_{n\in N_i} bw(d(n))} \times bw_i \quad (1)$$

In most cases, this assumption is just a rough approximation and not exact, but it is a starting point for the algorithm to work with. With further iterations the value will be gradually corrected.

The result of calculation (1) represents the bandwidth a sending vNet entity M_i could provide for a transfer $L_{i\to j}$ to a target M_j at maximum at time $t = 0$. The same value then has to be calculated for the receiving vNet object. The speed used for the transfer is then determined by the smaller of both values:

$$s_0(L_{i\to j}) = \min(s'_0(L_{i\to j}), s'_0(L_{j\to i})) \quad (2)$$

For the next steps ($t > 0$) the speed of a vTransfer instance will no longer be based on the predefined bandwidth of the involved vNets but on the previously calculated transfer speeds. The equations to determine the speed s of a transfer $L_{i\to j}$ at any time $t > 0$ are adjusted to:

$$s'_t(L_{i\to j}) = \underbrace{\frac{s_{t-1}(L_{i\to j})}{\sum_{n\in N_i} s_{t-1}(n))}}_{A} \times bw_i \quad (3)$$

$$s_t(L_{i\to j}) = \min(s'_t(L_{i\to j}), s'_t(L_{j\to i})) \quad (4)$$

With the factor A always being less or equal than 1.0 the maximum bandwidth of source network interface is never exceeded. Further A can only be equal to 1.0 in the case of a single active transfer, which results in that transfer using the maximum available capacity.

Using this algorithm we are able to determine the speed of a transfer at any time simply by factoring in either previously calculated transfer speeds in case of $t > 0$ or the neighboring machines theoretically possible transfer speeds in case of $t = 0$. A sample calculation for the example discussed above is shown in Table I. The first fours rows of the tables contain the values for s'_t for the transfer corresponding transfer expressed by the columns. The last row of each table contains the value of s_t. One can see that over time, the speeds of the transfers converge to a final value that overall saturates the vNets A to D.

TABLE II. PROPERTIES OF A WORKLOAD OBJECT.

Resource type	data type	description
vCPU	integer	Value expressing the total amount of calculation power the workload will use expressed in *cpu cycles*.
vMemory	f(cpu cycles)	Function returning the amount of used vMemory when a given amount of cpu cycles has be processed or is remaining.
vStorage, vNetwork	list	List of triplets expressing the amount of *cpu cycles* to be remaining or processed for a defined operation to be executed and a flag allowing parallel execution.

Finally, when using the model above, the calculated transmission speeds will always be based upon ideal network connections transmitted over ideal network cables without any kind of noise. Thus, we want to point out, that in our implementation we add in a factor called 'scatter' to express some randomness in latency and bandwidth and represent a combination of all kind of complex signal fluctuations. Our test show, that a variation of a few percent in transfer speeds is realistic due to physical connections not performing perfectly at all times.

IV. WORKLOAD DESCRIPTION

In our model, each workload is associated with a certain amount of load related to the resources described in Subsection III-A. We do not consider workloads not associated with any resource consumption, so the minimal defined workload will be a workload only consuming a single cycle of calculation power. This enables the use of *calculation power* as a reference for other resource usages: Instead of modeling usage of vMemory based on the time passed since the launch of the workload we model memory conumption as a function of consumed *calculation power*. In general, the definition of memory as a function over cpu cycles instead of processing time is a more realistic approach: a started but not advancing (i.e., due to missing resources) workload will not change its state and thus not change its consumption of memory.

Since the usage of storage and network are time-consuming processes due to the non-instant character of the underlying resource providers we cannot transfer this mechanism par for par. As a solution we designed storage and network usage as triggers likewise based on the amount of cpu cycles. Those triggers are simple integers and are executed as soon as the number of consumed or remaining cpu cycles has reached the configured value. Upon execution the trigger can then instantiate new network transfers or storage operation which are then processed. In addition the user can specify if those operations should be processed in parallel or if the workload should pause meanwhile.

Beside the described resource-related properties which are summarized in Table II each workload may execute user-defined subroutines upon reaching certain states to control the flow of workloads. Predefined states are *init* which gets executed when the workload is launched on a vMachine instance, *finish* which is associated with the workload terminating and

error, the state that a workload changes to upon failing. In addition a user can add other trigger and subroutines.

To start the simulation of a workload on a given infrastructure, the workload object has to be registered at a vMachine instance. The vMachine then enables access to its resources by reading the workload definition at the current simulation time and passing appropriate requests to the resource providers which then schedule the incoming requests.

V. IMPLEMENTATION

We realized the described model using Python 2.7 based on the concepts of the object-oriented programming paradigm. The most important components to mention are the different resource providers vNet, vMemory, vStorage and vNetwork along with vTransfer, the higher level entities vMachine and vDatacentre and finally the global environment object called *env*. Whereas most of the implementation can be considered straight forward, we want to highlight some aspects we consider implementation specific.

First, we implemented the resource provider's scheduling in case of more than one workload accessing the resource at a time is done in a round-robin manner. Hereby, the resource provider allocates an equal amount of the available resource to each consumer. In the future, we plan to implement other scheduling mechanisms.

Python natively does not offer an event handling concept, so we based our simulator on generators. A generator function allows to define a function that behaves like a iterator. The details of this concept are described in [15] (PEP0289 and PEP0342). This enables the implementation of time-dependent actions, like network transfers and storage operations as loops. An action requiring more than one time step to be performed is expressed as a set of smaller parts that are one time step long and yield at the end of their execution. We believe that using generators leads to less error-prone code due to the simplicity and the linear, synchronous layout.

Finally, due to the simulator not running in parallel but calling subtasks subsequently one at a time, one has to make sure that one task doesn't change the simulation state prematurely. We solved this by deep-copying the simulator's state before the execution of the subtask begins. Then, all subtasks refer to the copy when reading values while making the changes to the original state.

VI. CALIBRATION AND VALIDATION

To use the simulation framework, users must specify the available infrastructure by defining resource providers as explained in Section III-A. The model can be calibrated based on either of two correlated degrees of freedom: either the resource provider's performance or the workload's resource usage can be tuned. As soon as one dimension is set, the other has to be picked accordingly to get consistent results.

With the model being a simplification of reality it is up to the user to configure the infrastructure in a suitable way for one's use-case. Any calibration settings are workload specific, since for example a workload might or might not benefit from certain CPU features (i.e., Streaming SIMD Extensions (SSE) [16]) or network related characteristics (e.g., delays due to encryption settings).

TABLE III. AVERAGE PERFORMANCE OF COMMON CPUS REGARDING PSEUDO-RANDOM NUMBER GENERATION.

CPU name	clock frequency	Performance [MB/s]
Intel i3-2100	3.1 GHz	$16.3 \pm 0.9\%$
Intel i5-4288u	2.6 GHz	$13.5 \pm 1.4\%$
Intel Xeon E5520	2.27 GHz	$7.2 \pm 1.8\%$
Intel Xeon E5-2650	2.0 GHz	$3.6 \pm 8.5\%$

In addition to this systematic calibration, performance variations on all sorts of resources might occur due to non-perfect behavior of hardware. Those variations are comparably small and are covered by the aforementioned scatter factors we added to the model in our implementation. Those aspects are usually only known to system or software engineers or can be obtained by benchmarking.

We provide consistent calibration values based on the following assumptions for an Amazon EC2 environment:

- A workload that is modeled for a resource provider that supports a certain CPU feature may need less CPU cycles and is modeled accordingly.
- The same application that is modeled for a resource provider that does not support a certain CPU feature may need more CPU cycles and is modeled accordingly.
- The simulated execution of the same application using resource providers that are n-times more powerful than the original resource providers for which the application had been tuned in the beginning should take in the order of a n-th of the time when compared to the initial simulation.
- The simulated execution of an application should yield approximately the same results when compared to real run-time, respectively should be faster or slower in the same order of magnitude relative to the application the model's parameters were derived from.

To derive calibration values, we ran a high number (approx. 560,000) of tests on different machine configurations using a distributed benchmark suite. A detailed description of the tests can be found in [13]. To obtain the calibration values, we correlated the resources of the simulation environment with the performance of the corresponding real hardware.

A. Calibration

In a first step, we calibrated calculation power based on data of a synthetic CPU benchmark. In our case this is the single-threaded calculation of pseudo-random numbers for checksumming purposes on virtual machines of the EC2 instance type "t2.small" in the region "us-east-1". For our calibration, we set $2{,}000\,\mathrm{cpu\ cycles} \mathrel{\hat{=}} 2{,}000\,\mathrm{MHz}$, which is the average clock frequency a virtual CPU on the used EC2 instances provides as stated by Amazon. The benchmarks showed an average random number generation rate of $3.6 \pm 8.5\%$ megabytes per second per core using an Intel Xeon E5-2650 running at 2 GHz. Table III shows the results of the same benchmark on other CPUs for reference. Using this data, we can state that a real-world workload using the same amount of computing power as required for the generation of $3{,}600{,}000$ pseudo-random numbers would run one second. As mentioned above, this value might seem to be abstract compared to e.g.,

TABLE IV. AVERAGE PERFORMANCE OF COMMON CPUS REGARDING PSEUDO-RANDOM NUMBER GENERATION.

Evaluated transfer	distance	std. deviation
us-east-1 ↔ us-west-1	3,600 km	13.4%
us-west-1 ↔ eu-west-1	8,000 km	14.5%
ap-southeast-1 ↔ eu-west-1	11,300 km	17.3%
ap-southeast-1 ↔ us-west-1	13,900 km	19.7%

GFLOPS, but since a user might not necessarily know the amount of FLOPS his specific workload conducts as well as he might not know what CPU features are beneficial to his program, the description based on the benchmark results of his actual workload is more usable.

In contrast to the calibration of calculation power, storage and memory are less complex. The only free parameter associated with a vStorage instance is the bandwidth per time step. Nearly 50,000 performance tests on the aforementioned Amazon EC2 instances resulted in an average of $40 \pm 2.5\%$ MB/s. For vMemory, the only free parameter is the capacity, which is expressed as an integer. Other parameters, like e.g., access latency are not covered by the model.

The last parameters to be calibrated are those related to the network simulation. The bandwidth of a network device is primarily defined by its specifications and can usually be set straight forward. However, due to our calibration environment being virtualized, we had to measure the actual network capabilities which resulted in a bandwidth of 350 Mbit/sec±18%. Secondarily, the model considers the distance between sender and receiver as an influencing factor. A network transfer will be slower proportionally to the distance between two communicating machines. This factor can be significant and results in variations of transfer speeds of up to 13% (for transfers between us-east-1 situated in North Virginia, USA and us-west-1 in California, USA) to 20% (for transfers between us-west-1 and ap-southeast-1 in Singapore). A more detailed listing of the influence of distance between transfer endpoints on the fluctuation range of transfer speeds can be found in Table IV. We assume a fluctuation of 2% per kilometer on the first 4,000 km and 1% per kilometer above. This is in good accordance with the measured results and, in rough approximation, expresses that connections on the first 4,000 km are most likely connections on mainland with delays caused by e.g., routing devices. Longer distances can be assumed to be oversea connections that are mostly free of interference. The final parameter the model includes factors in the quality of the involved hardware components. Network transfers between the same machines on the same cable still fluctuate in a non-negligible, statistical manner following a gaussian curve. In our data we measured a standard deviation of 5%.

With the values deduced from our benchmarks and presented in this section, we want to validate the simulation environment's functionality by predicting the behavior of a real-world scenario using the simulator and compare the results.

B. Validation

The validation of the system's behavior regarding pure CPU load was done by simulating the calculation of 1,000 MB of pseudo-random numbers as explained in the calibration section. Based on the observations above, the expected runtime per job is 288 s. This expectation is covered by the simulation however, the observed values on real machines showed strong fluctuations of up to 31% and a standard deviation of 7.9 %. A detailed description can be found in [13].

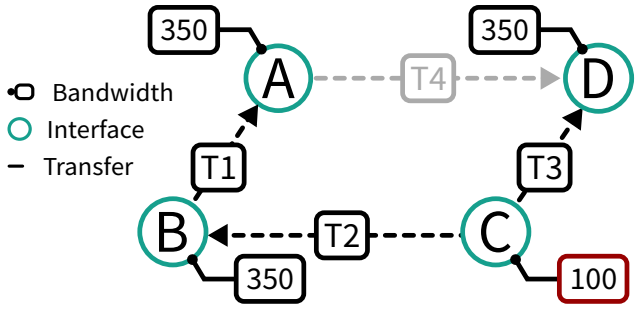

Figure 2. The graph representing the network validation environment.

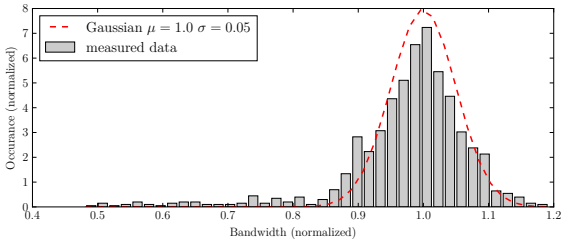

Figure 3. Normalized histogram of the measured bandwidths.

Next, we validated the behavior of the vStorage implementation. Since the system is based on a slow resource provider equal to the one implemented to represent network devices the validation of vStorage is shortened. The simulated workload is a single threaded application writing 10,000 MB to disk. According to the calibration, this process should take 250 s. We were able to reproduce this value in our simulation environment with a average deviation equal to the calibrated 2.5 %. This result was then compared to the performance values measured by `bonnie++`. We were able to show good accordance between the test cases although the aforementioned effect of varying performance biased the results. In numbers, the deviation between simulated and observed results for the duration of the benchmark was averaging at 5.3 % with peaks of 13.1 %.

Exemplary we choose a network setup similar to the one described in Section III-B and depicted in Figure 1 for validation of the network model. However, we slightly modified the scenario to match the environment we used for calibration (Amazon EC2, region *us-east-1*, "t2.small" instances). To be more conclusive, in addition to a static network layout, we assume another transfer between machine A and D to appear after five seconds. The new scenario is depicted in Figure 2 and reads as follows:

> At time $t = 0$, machine A initiates the transfer of 500 MB of data to machine B ($T1$). Meanwhile machine C sends 500 MB of data to B ($T2$) and 500 MB of data to D ($T3$). A, B, D hold 350 Mbit/s network devices, C is associated with a 100 Mbit/s device. At $t = 5$, A initiates an additional transfer of 500 MB of data to D ($T4$).

We executed the scenario 100 times in reality and using

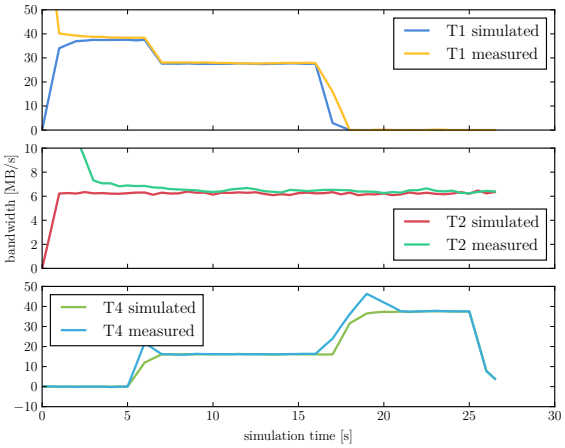

Figure 4. Comparison of measured and simulated bandwidths over time.

the simulator. In our first configuration, all machines are situated within the same data center and thus the geographical correction factor is zero and we only expect fluctuations of five percent due to our calibration. After dropping obvious outliers, the results confirm the assumption of a gaussian distributed micro-scattering as visualized in Figure 3.

The plot depicted in Figure 4 shows the transfer speeds for each transfer at a given time averaged over all performed benchmarks. Altogether we observed accurate accordance between simulation and reality as shown by the convergence of the transfer speeds to the expected values. However, in the first seconds, a large discrepancy can be seen. Since the values are far above the specifications of the network device, we assume the filling of caches or buffers being the cause. The transfers $T2$ and $T3$ evidently converge to $6.25\,\mathrm{MB/s}$, the theoretical maximum of the sending network card. $T1$ stays slightly below the theoretical maximum sending speed of A. This is expected since the overall bandwidth has to be shared with $T4$ as soon as $t \geq 5$.

In a second series of tests, we moved machine C to eu-west-1, $3{,}600\,\mathrm{km}$ away from the other machines. This leads to the geographical correction factor being relevant and thus to be factored in during calculation. Again, the whole scenario was done 100 times. This time we observed a bigger discrepancy and larger fluctuations. As mentioned in [13], factors like time of day respectively day of week do affect the performance of the virtual environments and their attached network devices. Since the deviation of the simulated results from those measured in reality is not larger than $7\,\%$ at any time $t_{\text{transfer}} > 3s$ during our tests, we consider the simplification as acceptable.

VII. Discussion, Conclusion and Outlook

As the tests show, our simulation framework provides good results after calibration has been done. The avarage deviation of the simulation results compared to real-world findings is lower than 7.9 percent for CPU simulation, lower than 5.3 percent for storage simulation and lower than 7 percent for network simulation.

The network simulation converges almost to the same mean transfer speed, only during the first few seconds the simulation underestimates the transfer speed that can be achived in the real-world. Our simulation never yields results for transfer speeds that are above the actual link speed - in contrast to the real world due to cache effects in the first few seconds. After around three seconds the simulated transfer rate is close to the real-world transfer rate. As the focus of our simulation framework is not to deliver a precise estimation of transfer speeds at all times, but to provide good overall results, the mentioned deviations are negligible in our case.

Our CPU simulations yielded very similar average results when compared to real-word tests. However, there are some deviations that can be explained by varying performance of the cloud infrastructure. As we found in [13], the performance of cloud infrastructure is not constant over time. Neglecting those variations, our results were quite good.

Our simulation framework allows to model infrastructure and workloads in a flexible way. There are two degres of freedom that allow to set one of them conviniently while choosing the other accordingly. The simulation framework yields consistent results and converges quickly. As it is written in Python, it is very lightweight and extensible with little effort. However, during testing we noted that some facts that are particularly important when simulating cloud infrastructure cannot yet be captured by our framework. Most importantly: cloud resources do not always yield the same performance. Without knowing the exact reasons, we strongly suspect that varying load of the providers' infrastructure as well as varying load of the inter-connecting networks.

Therefore, the next iteration of our simulation framework must support resource providers that yield a time-dependent amount of resources instead of a fixed number of, e.g., CPU cycles. Also, multi core support as well as the ability to capture CPU features are points to adress in future releases. Further we plan to expand the framework in the future to support a user when developing higher level functionalities such as scheduling mechanisms for distributed systems or simulations of platform services. Another aspect that is not yet covered by our framework and that should be tackled in future releases is a price model for infrastructure resources - something that is of particular interest in cloud computing.

References

[1] P. Mell and T. Grance, "The nist definition of cloud computing," National Institute of Standards and Technology, vol. 53, no. 6, 2009, p. 50.

[2] S. NIST, "800-145,"," A NIST definition of cloud computing",[online] http://csrc. nist. gov/publications/nistpubs/800-145/SP800-145. pdf, 2011, [accessed 1-February-2016].

[3] Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V., "Cloud Computing – Ein ganzheitlicher Blick über die Technik hinaus," 2010.

[4] BITKOM Research, "Cloud monitor 2014," 2014.

[5] S. hwan Lim, B. Sharma, G. Nam, E. K. Kim, and C. R. Das, "Mdcsim: A multi-tier data center simulation, platform," in in Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on, 2009.

[6] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and Experience, vol. 41, no. 1, 2011, pp. 23–50.

 ISBN: 978-1-61208-460-2

[7] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities," Jul. 2009, [accessed 1-February-2016].

[8] S. Garg and R. Buyya, "Networkcloudsim: Modelling parallel applications in cloud simulations," in Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on, Dec 2011, pp. 105–113.

[9] X. Li, X. Jiang, K. Ye, and P. Huang, "Dartcsim+: Enhanced cloudsim with the power and network models integrated," in Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on, June 2013, pp. 644–651.

[10] B. Wickremasinghe, R. Calheiros, and R. Buyya, "Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications," in Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on, April 2010, pp. 446–452.

[11] A. Núñez and et al., "icancloud: A flexible and scalable cloud infrastructure simulator," J. Grid Comput., vol. 10, no. 1, Mar. 2012, pp. 185–209.

[12] G. Castane, A. Nunez, and J. Carretero, "icancloud: A brief architecture overview," in Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on, July 2012, pp. 853–854.

[13] P. Krauss, T. Kurze, and A. Streit, "Cloudbench – A Framework for Distributed, Self-organizing, Continuous and Topology-aware IaaS Cloud Benchmarking with Super-peer Networks," in e-Science (e-Science), 2015 IEEE 11th International Conference on, Aug 2015, pp. 273–278.

[14] Wikipedia, "Max-Flow-Min-Cut-Theorem — Wikipedia, the free encyclopedia," 2015, [accessed 1-February-2016]. [Online]. Available: https://de.wikipedia.org/wiki/Max-Flow-Min-Cut-Theorem

[15] D. Goodger and B. Warsaw, "Index of Python Enhancement Proposals," 2016, [accessed 1-February-2016]. [Online]. Available: https://www.python.org/dev/peps/

[16] R. Ramanathan, R. Curry, S. Chennupaty, R. L. Cross, S. Kuo, and M. J. Buxton, "Extending the World's Most Popular Processor Architecture," Intel Corporation, White Paper, 2006.

 ISBN: 978-1-61208-460-2