How to Operate Container Clusters more Efficiently?

Some Insights Concerning Containers, Software-Defined-Networks, and their sometimes Counterintuitive Impact on Network Performance

Nane Kratzke, Peter-Christian Quint

Lübeck University of Applied Sciences
Department for Electrical Engineering and Computer Science
Center of Excellence CoSA
Germany
Email: {nane.kratzke, peter-christian.quint}@fh-luebeck.de

Abstract—**In previous work, we concluded that container technologies and overlay networks typically have negative performance impacts, mainly due to an additional layer to networking. This is what everybody would expect, only the degree of impact might be questionable. These negative performance impacts can be accepted (if they stay moderate), due to a better flexibility and manageability of the resulting systems. However, we draw our conclusion only on data covering small core machine types. This extended work additionally analyzed the impact of various (high core) machine types of different public cloud service providers (Amazon Web Services, AWS and Google Compute Engine, GCE) and comes to a more differentiated view and some astonishing results for high core machine types. Our findings stand to reason that a simple and cost effective strategy is to operate container cluster with highly similar high core machine types (even across different cloud service providers). This strategy should cover major relevant and complex data transfer rate reducing effects of containers, container clusters and software-defined-networks appropriately.**

Keywords–Container; Cluster; Software-Defined-Network; SDN; HTTP; REST; Microservice; Overlay Network; Performance; Docker; Weave.

I. INTRODUCTION

In previous work [1], we investigated the performance impacts of container technologies like *Docker* [2] and overlay network solutions like *Weave* [3]. Both systems are typical type representants for container or overlay network solutions.

Container technologies and overlay network solutions are often mentioned in the same breath with the popular microservice approach [4]. The microservice architectural style is applied successfully by companies like Amazon, Netflix, or SoundCloud [4], [5]. This architecture pattern is used to build very large, complex and horizontally scalable applications composed of small, independent and highly decoupled processes communicating with each other using language-agnostic application programming interfaces (API). Therefore, microservice approaches and container-based operating system virtualization experience a renaissance in cloud computing. Especially, container-based virtualization approaches are often mentioned to be a high-performance alternative to hypervisors [6]. *Docker* [2] is such a container solution, and it is based on operating system virtualization. Recent performance studies show only little performance impacts to processing, memory, network or I/O [7]. That is why *Docker* proclaims itself a ”lightweight virtualization platform” providing a standard runtime, image format, and build system for containers deployable to any Infrastructure as a Service (IaaS) environment.

Furthermore, container technologies and overlay network solutions are often used under the hood by popular container cluster platforms like *Mesos* [8], *CoreOS* [9] or *Kubernetes* [10]. The reader is referred to more detailed analysis of such kind of platforms [11], [12]. These cluster solutions are often the technological backbone of microservice based and highly scalable systems of cloud deployed systems. Nevertheless, corresponding performance impacts of the underlying technologies have been hardly investigated so far. Additionally, distributed cloud based microservice systems of typical complexity often use hypertext transfer protocol (HTTP) based and representational state transfer (REST) styled protocols to enable horizontally scalable system designs [13].

Beside this, container clusters often rely on so called overlay networks under the hood. Because overlay networks are often reduced to distributed hashtable (DHT) or peer-to-peer approaches, this paper uses the term software-defined-networks (SDN). SDNs, in the understanding of this paper, are used to provide a logical internet protocol (IP) network for containers on top of IaaS infrastructures.

This study investigates the performance impact of one representative SDN solution called *Weave* [3] for *Docker* [2] and it is outlined as follows. Section II presents related work about state-of-the-art network benchmarking of container and container cluster approaches and SDN solutions. If container clusters are to be distributed across multiple cloud infrastructures, the selection of appropriate machines is not trivial. Section III focuses on this problem and shows how and why we have selected exactly six virtual machine types of two different cloud service providers (AWS and GCE) to do our benchmarking. Section IV explains the experiment design to identify performance impacts of containers and SDNs. The used benchmark tooling is provided online [14]. Resulting performance impacts are analyzed and discussed in Section V. Derived conclusions and insights to minimize performance impacts on application, overlay network and IaaS infrastructure layer are presented in concluding Section VI.

II. RELATED WORK

To handle complexity of common microservice approaches, software developers often have to design, use or adapt existing SDN software, that will run on commodity hardware. One of the main challenges of SDN is to reach the performance known from proprietary software running on purpose-built hardware. This contribution will provide some insights into this gap, dealing with influences of popular container technologies, cluster approaches and software-defined-networks focusing HTTP and REST-based network performance often used in microservice architectures.

A. Container technologies

Although container based operating system virtualization is postulated to be a scalable and high-performance alternative to hypervisors. Hypervisors are still the standard approach for IaaS cloud computing [6]. Felter et al. provided a very detailed analysis on CPU, memory, storage and networking resources to explore the performance of traditional virtual machine deployments, and contrast them with the use of Linux containers provided via *Docker* [7]. Their results indicate that benchmarks that have been run in a *Docker* container show almost the same performance (floating point processing, memory transfers, network bandwidth and latencies, block I/O and database performances) like benchmarks run on "bare metal" systems. Nevertheless, Felter et al. did not analyze the impact of containers on top of hypervisors like this contribution does.

Furthermore, the reader should be aware that *Docker* is not the only container solution. Almost all operating systems provide some lightweight virtualization mechanisms like *Docker* does for the Linux and further operating systems. So to some degree, all of our conclusions can be transferred to lightweight virtualization approaches shown in Table I.

If more than one container has to be operated with **high availability and scalability requirements** in mind, so called container clusters come into play. We will discuss these technologies in Section II-B.

B. Container Cluster

Container clusters are similar to compute clusters. But container clusters run containers instead of processing jobs. It is up to the container cluster (to some definable degree) to decide which resource is allocated to run a container. Container clusters like *Mesos* [8], *CoreOS* [9] or *Kubernetes* [10] can be deployed on thousands of machines (nodes). These nodes can be hosted on "real hardware" machines but also on virtual machine instances in private or public clouds. A container cluster can be even deployed across different cloud service providers and different IaaS infrastructures. This results in some positive benefits for operation. (Regional) failures can be compensated, overload can be distributed and vendor lock-in can be avoided [11], [15]. However, all provided machines for such kind of cluster should show **similar performance characteristics** to enable fine-grained resource allocation capabilities of a container cluster [16]. This can be tricky to achieve in multi-provider scenarios and we will show in Section III how this was considered for our machine type selection.

Furthermore, container clusters often use SDN solutions under the hood to **hide networking specifics** from IaaS cloud service provider infrastructures. This will be discussed in following Section II-C.

C. Software-Defined-Networking

Software-defined-networking (SDN) is a paradigm where a software program dictates the overall network. SDN brings benefits, especially in cloud computing. Because it is easier to change and manipulate than using a fixed set of commands in proprietary network devices, SDN enables a high degree of flexibility. Cloud computing enables to launch or terminate instances very fast and on demand. In this context, accompanying scalability, elasticity and flexibility requirements are hard to handle with legacy network platforms [17].

Although there exist several SDN solutions for *Docker*. Pure virtual local area network (VLAN) solutions like *Open vSwitch (OVS)* [18] are not considered, because *OVS* is not to be designed for operating system virtualization. Two common SDN solutions are *Weave* [3] and *Flannel* [19]. *Weave* and *Flannel* are using a similar technique to build an overlay network. *Flannel* is developed by *CoreOS* [9] and optimized for the same named operating system. It turned out that *Flannel* can not be installed frictionless on another operating system without far-reaching modifications to the system. So we decided to use *Weave* as representative SDN solution. *Calico* [20] is a pure layer 3 approach to virtual networking for highly scalable data centers but was out of our focus because it requires *Docker 1.9* to work seamlessly. *Docker 1.9* had been released after our benchmark analysis. However, *Calico* is promising and will be extended to our here presented solution in ongoing work.

Weave creates a network bridge on *Docker* hosts to enable SDN for *Docker* containers. Each container on a host is connected to that bridge. A *Weave* router captures Ethernet packets from its bridge-connected interface in promiscuous mode. Captured packets are forwarded over the user datagram protocol (UDP) to *Weave* router peers running on other hosts. These UDP "connections" are duplex and can traverse firewalls.

D. Benchmarking network performances in containerized environments

To analyze the performance impact of containers, software-defined-networks and machine types, this paper considered several contributions on cloud related network performance analysis [21], [22], [23], [24], [25], [26], [27]. So far, there exist little papers focusing container technologies (only [7] provide some data). But none of these contributions focused explicitly on horizontally scalable systems with HTTP-based and REST-like protocols. To address this common use case for microservice architectures, this paper proposes a special experiment design being described in Section IV.

So, there exist almost no special network benchmarks focusing containerized environments and microservice related system design questions. But of course, there exist several well established **TCP/UDP networking benchmarks** like *iperf* [28], *uperf* [29], *netperf* [30] and so on. [24] provide a much more complete and comparative analysis of network benchmarking tools. [31] present a detailled list on cloud computing related (network) benchmarks. Most of these benchmarks focus pure TCP/UDP performance [24] and rely on one end on a specific server component used to generate network load. These benchmarks are valuable to compare principal network performance of different (cloud) infrastructures by comparing

TABLE I. Lightweight virtualization mechanisms on various operating systems

Operating system	Virtualization mechanism	Further links (last access 9th Nov. 2015)
Linux	Docker	http://www.docker.io
	LXC	http://linuxcontainers.org
	OpenVZ	http://openvz.org
Solaris	Solaris Zones	http://docs.oracle.com/cd/E26502_01/html/E29024/toc.html
FreeBSD	FreeBSD Jails	http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/jails.html
AIX	AIX Workload Partitions	http://www.ibm.com/developerworks/aix/library/au-workload/index.html
HP-UX	HP-UX Containers (SRP)	https://h20392.www2.hp.com/portal/swdepot/displayProductInfo.do?productNumber=HP-UX-SRP
Windows	Sandboxie	http://www.sandboxie.com
	Hyper-V Container (planned)	http://azure.microsoft.com/de-de/blog/microsoft-unveils-new-container-technologies-for-the-next-generation-cloud/

what maximum network performances can be expected for specific (cloud) infrastructures. But maximum expectable network performances are in most cases not very realistic for REST-based protocols.

Other **HTTP related benchmarks** like *httperf* [32] or *ab* [33] are obviously much more relevant for REST-based microservice approaches. These tools can benchmark arbitrary web applications. But because the applications under test are not under direct control of the benchmark, these tools can hardly define precise loads within a specific frame of interest. Therefore, HTTP related benchmarks are mainly used to run benchmarks against specific test resources (e.g., a HTML test page). But this makes it hard to identify trends or non-continuous network behavior.

Our presented approach is more a mix of above mentionend benchmarking strategies of pure TCP/UDP benchmarks and HTTP benchmarks. Section IV will show that we use a benchmark frontend (which is conceptually similar to *httperf* or *apachebench*) and a reference implementation under test (*ping-pong* system, which is conceptually similar to a *iperf* server but designed to measure HTTP performance). Furthermore, most of the above mentioned benchmarks concentrate on data measurement and do not provide appropriate visualizations of collected data. This may hide trends or even non-continuous network behavior. To cover this, our presented benchmarking approach focus data visualization as well. Often cloud service provider specific influences are not considered systematically in recent studies. To consider such cloud service provider specific influences on network performance, we performed a very systematic virtual machine type selection for our experiments (see Section III).

III. Virtual Machine Type Selection

Selecting virtual machine types can be complex in its details. The underlying decision making problem makes it difficult to model it appropriately. There exist several complex mathematical models like analytic hierarchy processes [34], utility function based methodologies [35], outranking approaches [36], simple additive weighting approaches [37], cloud feature models [38], dynamic programming approaches [39], cloud service indexing [40], ranking approaches [41], Markov decision process [42] or even astonishing combinations of Fuzzy logic, evidence theory and game theory [43] to support this task. However, in order to establish a container cluster across multiple providers, it is necessary to select most similar machines [16]. None of the above mentioned approaches focuses similarity. All mentioned approaches try to identify a best resource in terms of performance or cost effectiveness. Furthermore, very often cloud service selection is not done very systematically in practice. Often, the decision for a specific cloud service provider infrastructure is more an evolutionary process than a conscious selection. At some point in time, there might arise the need to change a cloud service provider or to diverse a deployment across several cloud service providers. There exist several technologies to support this. One approach is to use container cluster technologies like introduced in Section II-B. Well known companies like Google, Netflix, Twitter are doing this very successful to handle regional workloads, to provide failover and overflow capacities. Small and medium sized companies can benefit as well. Nevertheless, these type of clusters rely on homogeneous nodes (nodes with similar performance characteristics). Otherwise, the intended fine-grained resource allocation of container clusters gets limited.

EasyCompare [31] is a tool with the focus on identifying most similar (not best performing or most cost effective) machine types across different IaaS cloud service providers. The suite uses a feature vector shown in equation (1) to describe the performance of a virtual machine type. EasyCompare uses several different benchmarks. **Processing performance** is determined by the **Taychon** benchmark [44]. The **Stream** benchmark is used to measure the **memory performance** [45]. With the **IOZone benchmark** [46] the read and write **disk performance** can be measured. iperf [28] is used to measure **network transfer rates** of intra cloud data transfers.

$$\mathbf{i} = \left(\begin{array}{c} i_1 \\ i_2 \\ \\ i_3 \\ i_4 \\ \\ i_5 \\ \\ i_6 \\ \\ i_7 \\ \\ \end{array}\right) \begin{array}{l} \text{Processing: Amount of simultaneous executable threads} \\ \text{Processing: Processing time in seconds} \\ \text{ (Median of all Tachyon benchmark runs)} \\ \text{Memory: Memory size in MB} \\ \text{Memory: Memory transfer in MB/s} \\ \text{(Median of all Stream Triad benchmark runs)} \\ \text{Disk: Data transfer in MB/s for disk reads} \\ \text{(Median of all IOZone read stress benchmark runs)} \\ \text{Disk: Data transfer in MB/s for disk writes} \\ \text{(Median of all IOZone write stress benchmark runs)} \\ \text{Network: Data transfer in MB/s via network} \\ \text{(Median of all iperf benchmark runs)} \end{array} \tag{1}$$

The similarity $s(\mathbf{i}, \mathbf{j})$ of two virtual machine types $\mathbf{i}$ and $\mathbf{j}$ can be analyzed by calculating their vector similarity. Although this approach is mostly used in domains like information retrieval, text and data mining, vector similarities can be used to determine the similarity of virtual machine types as well. A

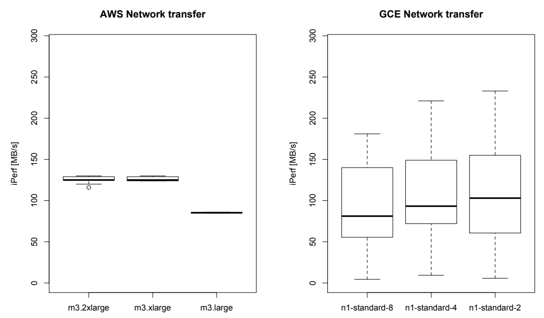


Figure 1. Boxplots of measured network performances using iperf on selected virtual machine types, taken from [31]

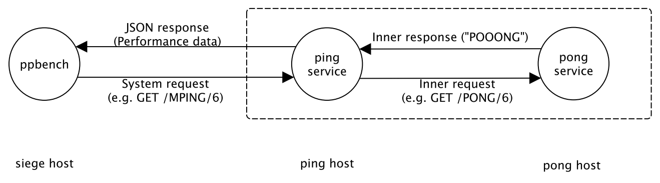
good similarity measure has to fulfill three similarity intuitions [47]:

- **Intuition 1:** "*The similarity between A and B is related to their commonality. The more commonality they share, the more similar they are.*"
- **Intuition 2:** "*The similarity between A and B is related to the differences between them. The more differences they have, the less similar they are.*"
- **Intuition 3:** "*The maximum similarity between A and B is reached when A and B are identical.*"

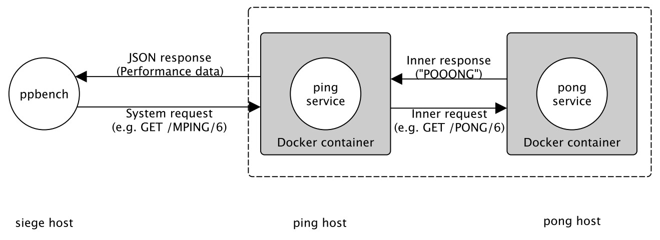
According to [31], it turned out that the normalized euclidian distance based shown in equation (2) measure fulfills Lin's intuitions [47] of a good similarity measure for the intended purpose of identifying most similar cloud resources across different providers. This variant of the Euclidian distance metric measures the relative performance relations of performance components $(i_1, i_2, \dots, i_m)$ and $(j_1, j_2, \dots, j_m)$ and normalizes the result $s(\mathbf{i}, \mathbf{j})$ to a similarity value between 0.0 ($\mathbf{i}$ and $\mathbf{j}$ are not similar at all) and 1.0 ($\mathbf{i}$ and $\mathbf{j}$ are completely similar).

$$\begin{aligned}
\forall \mathbf{i}, \mathbf{j} \quad & s(\mathbf{i}, \mathbf{j}) = 1 - \frac{1}{m} \sum_{n=1}^{m} \left(1 - \frac{\min(i_n, j_n)}{\max(i_n, j_n)}\right)^2 \\
\forall \mathbf{i}, \mathbf{j} \quad & s(\mathbf{i}, \mathbf{j}) = s(\mathbf{j}, \mathbf{i}) \\
\forall \mathbf{i}, \mathbf{j} \quad & 0 \leq s(\mathbf{i}, \mathbf{j}) \leq 1 \\
\forall \mathbf{i} \quad & s(\mathbf{i}, \mathbf{i}) = 1
\end{aligned} \tag{2}$$

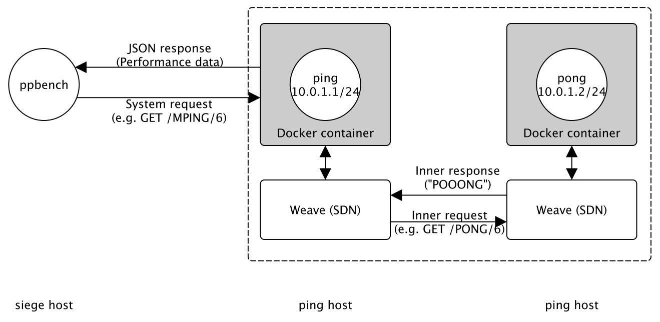
The reader is referred to Appendix Table IV. This compares machine types provided by two major and representative public cloud service providers: Amazon Web Services and Google Compute Engine. Finally, in over 500 different possible counterpart pairings (see Table IV) only three machine pairings have a strong similarity of almost 1. These pairings are reasonable choices for container clusters, and that is why we chose them as objects of investigation. So the AWS instance types m3.2xlarge, m3.xlarge, m3.large and GCE machine types n1-standard-8, n1-standard-4 and n1-standard-2 are used for the determination of performance impacts when using *Docker* and SDN technologies. Their measured network performance is shown in Figure 1.



(a) **Bare** experiment to identify reference performance



(b) **Docker** experiment to identify impact of container



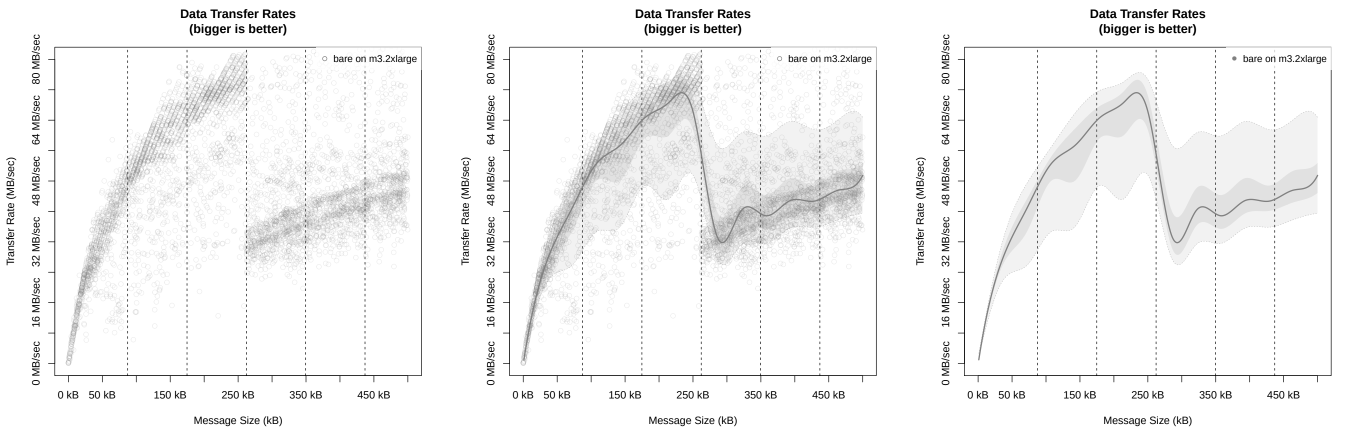
(c) **Weave** experiment to identify impact of SDN

Figure 2. Experiment setups

IV. EXPERIMENT DESIGN

To analyze the network performance impact of container, SDN layers and machine types on the performance impact of distributed cloud based systems using HTTP-based REST-based protocols, several experiments have been designed (see Figure 2). The analyzed *ping-pong* system provides a REST-like and HTTP-based protocol to exchange data. This kind of connections are commonly used in microservice architectures and container based cloud systems. The *ping* and *pong* services were developed using Googles *Dart* programming language [48]. It is possible for the siege to send a request to the *ping-pong* system. Via this request the inner message length between *pong* and *ping* server can be defined. So, it is possible to measure HTTP based transfer rates and answer times between *ping* and *pong* for a specific message size.

In our initial contribution [1] we installed *Apachebench* on the siege server [49] to collect performance data of the *ping-pong* system. But with high-core virtual machines we identified several network connection problems, which almost did not occur on low core machines like (AWS m3.medium and m3.large, GCE n1-standard-2). Especially in the range of message sizes of about 250kB we identified a lot of network connection errors between *ping* and *pong* host on high core machines (the effect was less severe or did not occur at all with smaller and bigger message sizes). While



(a) Plot of measurements

(b) Plot of measurements with median line and 90% and 50% confidence intervals

(c) Plot of median line and confidence intervals only, still enables to estimate skewness of distribution while reducing jitter in presentation

Figure 3. Different visualizations of the same data by example transfer rates measured on a m3.2xlarge instance

it was not clear of first, what causes these phenomenon, it turned out that this was due to flooding connection requests at a critical point that relates with the standard TCP receive window size configured on *ping* and *pong* host (see Section V-B for analysis). This effect did not occur with single (like m3.medium[1]) or even double core machines. But on high core machines *Apachebench* pushed the system into oversaturated network conditions producing useless benchmark data. That is why we decided to adapt our experiment setting to some degree.

Instead of *Apachebench* we decided to run our experiments with a special developed benchmarking script (*ppbench*). *Ppbench* sends several HTTP request with a random message size m between a minimum and maximum amount of bytes to the *ping* server to analyze the answer and response behaviour for different message sizes. For this contribution we covered the message sizes between 1 byte and 500kB. *Ppbench* was used to collect a 10% random sample of all possible message sizes. The *ping* server relays each request to the *pong* server. And the *pong* server answers the request with a m byte long answer message (as requested by the HTTP request). The *ping* server measures the time from request entry to the point in time when the *pong* server answer hit the *ping* server again. If there are connection problems the *ping* server retries to connect the *pong* server for a specified amount of times. If the *ping* server can not connect with the *pong* server it answers with a HTTP 503 code. The *ping* server answers the request with the measured time between *ping* and *pong*, the length of the returned message, the HTTP status code send by *pong* and the number of retries to establish a HTTP connection between *ping* and *pong*. *Ppbench* repeats a request for a specified message size m several times (in our case 10 times) to calculate a mean transfer rate. Using this approach we got a much more complete coverage and insight into our analyzed problem domain.

The **Bare experiment setup** shown in Figure 2(a) was used to collect reference performance data of the *ping-pong* system deployed to different virtual machines interacting with a REST-like and HTTP based protocol. Further experiments added a container and a SDN solution to measure their impact on network performance. A *ping* host interacts with a *pong* host to provide its service. Whenever the *ping* host is requested by *siege* host, the *ping* host relays the original request to the *pong* host. The *pong* host answers the request with a response message.

The intent of the **Docker experiment setup** was to figure out the impact of an additional container layer to network performance (Figure 2(b)). So, the *ping* and *pong* services are provided as containers to add an additional container layer to the reference experiment. Every performance impact must be due to this container layer.

The intent of the **Weave experiment setup** shown in Figure 2(c) was to figure out the impact of an additional SDN layer to network performance. *Weave*, a representative SDN solution, connects the *ping* and the *pong* containers. So, every data transfer must pass *Weave* between *ping* and *pong*. Therefore, every performance impact must be due to this additional networking layer.

V. DISCUSSION OF RESULTS

To generate statistical significant data, our benchmarking tool generates for the analyzed space of message sizes (1 byte upto 500kB) something about 5000 data points. Each benchmarked data point aggregates 10 single measurements. And this was done for six machine types of two providers several times. In other words, we have a lot of data points to present (and to confuse the reader). We will explain our presentation of this data in Section V-A. Section V-B describes an astonishing non-continuous behavior of data transfer rates, which will guide us contemplating about container impact in Section V-C and SDN impact in Section V-D.

A. Presentation of data

All relevant statistical data processing and data presentation has been done by statistical computing toolsuite R [50]. We decided to present the data per machine type as transfer plots

[1]This was the machine type we used for our initial publication [1]

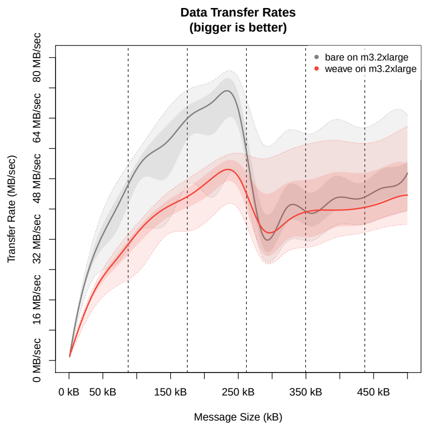
(to visualize the changing data transfer rates of different message sizes) and loss plots (to visualize the relative performance change of a data series compared with a reference data series). All transfer plots can be found in Figure 5, all loss plots can be found in Figure 6. In the following subsections we explain how these plots have been generated and how these plots have to be read by the reader.

1) Transfer plots: If we would barely plot our measurements, we would get a graphical result like in Figure 3(a). This kind of presentation provides a good and realistic impression about the distribution of measurements but without guiding descriptive statistical data. To draw general conclusions, we should not be interested in single data points but in trends and confidence intervals. Figure 3(b) shows exactly the same dataset but with a (spline smoothed) median line and 50% (indicating all values between the 25% and 75% percentile) and 90% confidence intervals (indicating the range of all measured values between the 5% and 95% percentile) as an overlay. This helps to reason about the skewness of a distribution and to handle outliers statistical appropriate. So Figure 3(b) provides the most information. But if we compare two or three series of (partly overlaying) data, it should be obvious for the reader that to plot all data points would mean a lot of confusion. The reader would get lost in details of thousands of data points. That is why we have decided to plot only the descriptive statistical data (median, 50% and 90% confidence interval) if we compare two or more data series. However, the reader should be aware that the reduced presentation of data – like done exemplarily in Figure 3(c) – still is based on a statistical significant amount of data points like shown in Figure 3(a). This is true for all data presentations shown in Figure 5. But for a better readability we decided not to plot the data points in comparison plots.

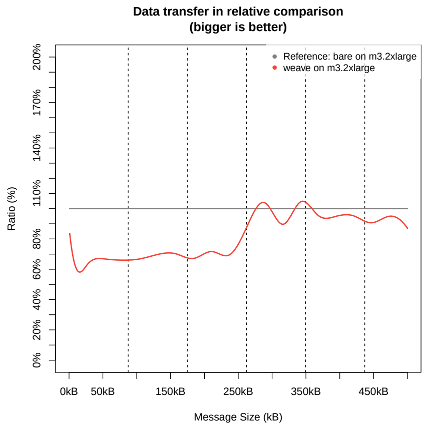
2) Loss plots: Transfer plots as shown in Figure 4(a) enable us to compare two or more data series by looking at their descriptive statistics of their absolute values. But to compare data series in a relative way is more useful, especially if we consider that machine types can be provided by different cloud services providers (in our case AWS and GCE). The reader may consider Figure 1 to see that AWS and GCE provide similar but not identical data transfer rates in their infrastructures for specific virtual machine types. Even the variances may differ substantially; AWS infrastructure shows almost no variances (very small box plots), but GCE shows substantial variances in data transfers). So, it is not helpful to compare absolute values to reason about performance impacts of container and SDN solutions in different IaaS infrastructures and for different virtual machine types due to different absolute levels of network performances and differing variances.

Therefore, we provide additional plots (see Figure 4(b)), to visualize the relative performance impact of containers or SDN compared with a reference experiment (in our case this is always the bare experiment on a specific machine type). A loss line is simply the division of two median lines (the two median lines $red/grey$ shown in Figure 4(a)).

$$
\begin{aligned}
loss_{exp,mach}(m) &= \frac{trans_{mach,exp}(m)}{trans_{mach,\text{bare}}(m)} \\
mach &\in \{\text{m3.large}, ..., \text{n1-standard-8}\} \\
exp &\in \{\text{bare}, \text{docker}, \text{weave}\}
\end{aligned}
\tag{3}
$$



(a) Exemplary absolute comparison of two data series



(b) Exemplary relative comparison of two data series (same data set as above)

Figure 4. Absolute and relative comparison of two datasets by example of m3.2xlarge bare and *Weave* experiment data

And of course the loss of $loss_{\text{bare},mach}(m) = 1$. That is the grey reference line in Figure 4(b). All loss plots of all analyzed machine types are presented in Figure 6.

B. TCP receive windows define small and big messages

While transfer plots and loss plots are helpful to identify general trends, it is worth to come back to the bare plots like shown in Figure 3(a) exemplarily. The reader can identify a sharp break of transfer rates at about 250kB. This is exactly the same range where *Apachebench* produced a lot of network errors for high core machine types. We can measure transfer rates up to a specific transfer rate for messages smaller 250 kB. But these transfer rates are almost halfed abruptly for message sizes greater than 250 kB for a substantial amount of measurements (but not for all). It turned out that this effect could be measured for all analyzed machine types (see Appendix Figure 7) and in both IaaS infrastructures (AWS, GCE). So, it is not due to some traffic shaping caused by the IaaS infrastructure. Furthermore, the effect occurred exactly at three times the standard TCP receive window size (87380 bytes) of the used Ubuntu Linux 14.04 systems under test. The reader is referred to RFC 1323/7323 [51], [52] if being unfamiliar with TCP window sizes. That is why we plot all multiples of the TCP standard receive window in our plots (dashed lines) to provide the reader some visual guidance of this effect.

We are quite sure that this has to do with the TCP protocol stack and/or a combination with the used *Dart* language and its HTTP library. For investigation and cross-checking we implemented the *ping-pong* system in *Java* as well and used the standard *Java* HTTP library (com.sun.net.httpserver). It turned out that the *Java* implementation of the *ping-pong* system does not show exactly the same effect but something similar. The *Java* curve reached its peak (and stayed on that level) at about three times the TCP standard receive window size, it stays below the *Dart* curve in the second and third TCP standard receive window but showed slightly better results in the first, fourth and fifth receive window. Finally, in the six window it showed the same performance as the *Dart* implementation. Furthermore, the *Java* implementation showed some significant outliers to the bottom exactly around the TCP standard receive window. In total (from first to sixth TCP standard receive window) the *Dart* implementation showed better results than the *Java* implementation. So Google seemed to optimized its *Dart* HTTP library at the cost of non-linear behavior. So, whatever the cause for this effect is, it seems have something to do with the TCP stack. However, the effect enables us to define a clear criterion to define small and big message sizes (for *Dart* and other languages as well). And the reader will see that containers and SDNs behave different for small and big messages sizes.

Thus, we define **small messages** and **big messages** as messages smaller/bigger than three times (for *Dart*, for *Java* and other languages this would be another value) the **TCP standard receive window size** of a system.

In other words, small and big messages are operating system and virtual machine specific but configurable. The TCP standard receive window size is configurable on Linux systems for instance.

C. Container impact

The container impact for all machines can be seen in Figure 6 (all red lines). Table II summarizes these figures to some handsome guidance levels (and, therefore, not exact numbers). We see that the container impact is responsible to reduce the network transfer rates down to about 90% for small message sizes. But we although notice an astonishing effect. Transfer rates are **increasing** for big message sizes. The transfer rates are increased up to 120% to 130% for 4 and 8 core machines. That means an additional layer improves the overall data transfer performance (for big messages). This effect is hardly measurable for small core machines. Containers enlarge the network stack and therefore introduce more buffering capacities. This is obviously of minor relevance for small messages but seems to be positive for big messages.

D. SDN impact

The impact of SDNs for all machines can be seen in Figure 6 (all blue lines). Table III summarizes these figures to some handsome guidance levels (and, therefore, not exact numbers). The SDN impact can be much more severe than the container impact. This is especially true for small core virtual machine types like m3.large (AWS), n1-standard-2 (GCE). This has to do with the fact that SDN software routers contend for the same (and very limited) CPU with processing services. The worst case we have measured is for the n1-standard-2 GCE virtual machine type, where this effect is responsible to reduce the network transfer rates to about 25% of the bare reference system.

For 4-core machine types this effect is getting minimized and transfer rates are going down to about 60% of the bare reference system. For 8-core machine types this loss can be reduced to 70% of the bare reference system for small messages.

And again we see an astonishing effect for big messages. Transfer rates are **recovering** for big message sizes. On the AWS infrastructure the data transfer performance of m3.xlarge and m3.2xlarge instances is hardly distinguishable from the bare reference experiment for big messages. The GCE infrastructure seems to be more susceptible and is recovering less extensive for big messages. However, it is recovering! This might have to do with a general higher data transfer variance, which can be observed in the GCE infrastructure (see Figure 1). Like containers, SDNs enlarge the network stack and, therefore, introducing much more buffering capacities. This results in negative performance impacts for small messages but seems to be positive for big messages on high core virtual machine types.

E. Summary

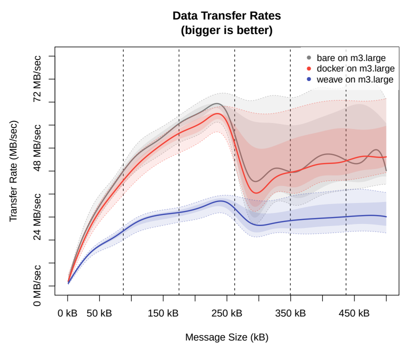
In our initial publication, we concluded that container and SDN solutions reduce network performance due to enlarging network stacks (this effect could be measured on single core machines very well). But these solutions are inducing more buffering capacities along an extended network stack as well. So, containers and SDNs are accompanied by positive and negative effects at the same time. In fact, it turned out, that an additional container layer can even improve the overall network performance for high core machine types. SDN solutions can have severe impacts. This is especially true for low core virtual machine types. However, on high core machines, the impact is moderate and for big message sizes SDN solutions might be even hardly measurable in data transfer rates (see Figure 6(c,e)).

VI. CONCLUSION AND OUTLOOK

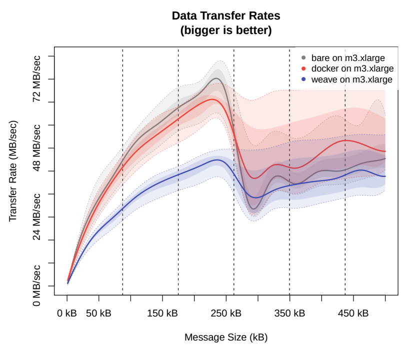
To use HTTP-based, REST-like containerized services is a common approach for modern horizontally scalable and cloud

TABLE II. Relative performance impacts of an **additional container layer** (*Docker*) to data transfer rates; just guidance levels; please check Figure 6 (red lines) for more details

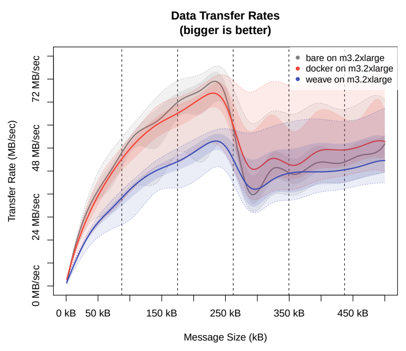
Machine type	cores	Small messages	Big messages
AWS m3.large	2	90% - 100%	100% - 110%
GCE n1-standard-2	2	95%	95% - 100%
AWS m3.xlarge	4	90%	120% - 130%
GCE n1-standard-4	4	95%	120% - 130%
AWS m3.2xlarge	8	90%	110% - 130%
GCE n1-standard-8	8	90%	110% - 120%



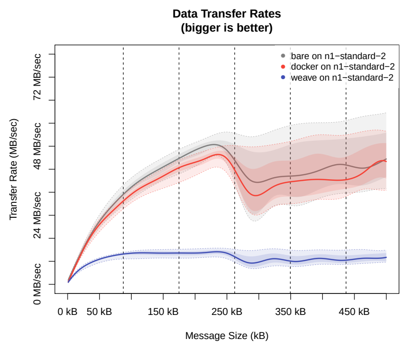
(a) AWS m3.large (2 core)



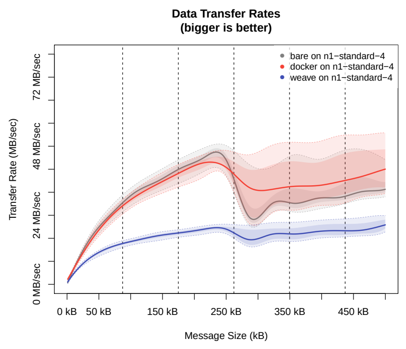
(b) AWS m3.xlarge (4 core)



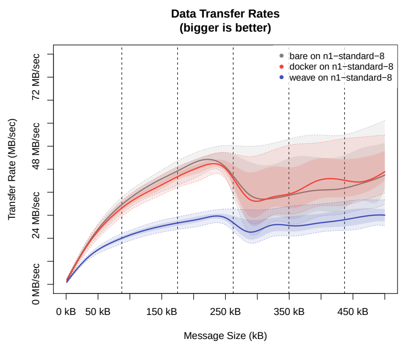
(c) AWS m3.2xlarge (8 core)



(d) GCE n1-standard-2 (2 core)



(e) GCE n1-standard-4 (4 core)



(f) GCE n1-standard-8 (8 core)

Figure 5. Median transfer rates with 90% confidence bands (5% and 95% percentile) and TCP standard receive window of 83780 bytes (dotted lines)

TABLE III. Relative performance impacts of an **additional SDN layer** (*Weave*) to data transfer rates; just guidance levels; please check Figure 6 (blue lines) for more details

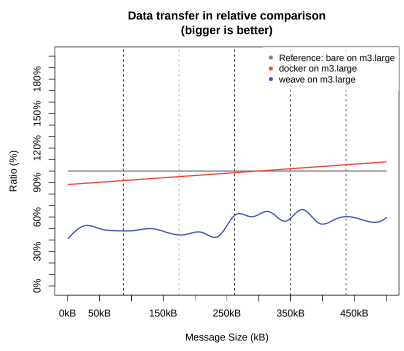
Machine type	cores	Small messages	Big messages
AWS m3.large	2	45%	60%
GCE n1-standard-2	2	25% - 80%	25%
AWS m3.xlarge	4	60%	90% - 100%
GCE n1-standard-4	4	40% - 100%	60%
AWS m3.2xlarge	8	70% - 80%	90% - 100%
GCE n1-standard-8	8	55% - 80%	70%

deployed microservice based systems. These technologies are not intentionally used to make cloud systems more performant but to make their operations and deployment more manageable. Astonishingly, we identified several cases where these technologies could even improve data transfer performances.

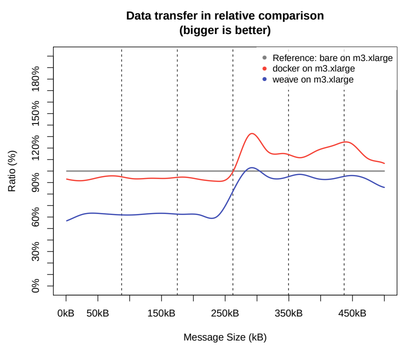
The impact of containers and SDNs on data transfer rates seems related with TCP standard receive windows (see RFC 1323/7323 [51], [52]). The window size allows a precise criterion what small and big messages are. Furthermore, we identified that small and big messages can show substantial differing transfer rates. This insight might be valuable for network optimizations of container clusters due to the fact, that TCP standard receive windows sizes can be configured by an operating system. However, this point stays open for ongoing research. In ongoing research we investigate further programming languages like Go, Ruby and Java to cover more programming languages and language specific behaviors because we identified that non-continuous network behavior seems to be deeply language (or library) specific. According to our preliminary results, almost every analyzed programming language (or their standard HTTP libraries) showed such non-continuous points.

For five of six analyzed machine types from AWS and GCE, we even identified a positive effect of containers on data transfer rates for big messages (see Table II) and we even identified a recovering effect on transfer rates for big messages due to SDN (see Table III). Furthermore, it is interesting to know that the performance loss of SDN for small messages is much more intensive on low core machines (in worst case on a GCE n1-standard-2 machine we only measured 25% of the reference performance) than on high core machines (about 80% of the reference performance).

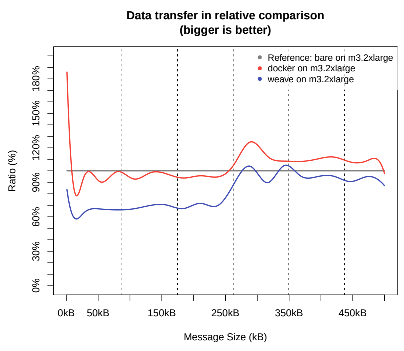
So far, the presented data has some shortcomings. We only presented data transfer rates, but the used benchmarking solution is able to generate other metrics like round trip latencies or requests per second as well. However, the derived insights are basically the same. So, we did not present these metrics here. Our systematic virtual machine type selection has been only applied to AWS and GCE but can be easily extended to other public cloud service infrastructures like Microsoft Azure or private cloud infrastructures like *OpenStack* or *Eucalyptus*. This is up for ongoing work. For pure pragmatic reasons, we only covered message sizes up to 500kB. Modern NoSQL databases can easily return much larger messages. However, we pushed our system under test into a saturated network condition. So we think, it is unlikely to get new insights



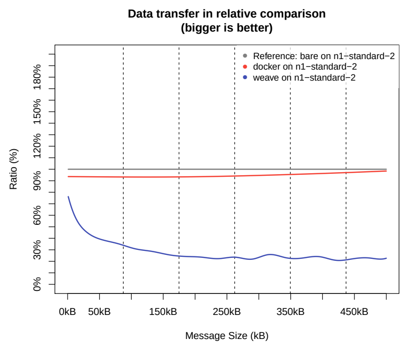
(a) AWS m3.large (2 core)



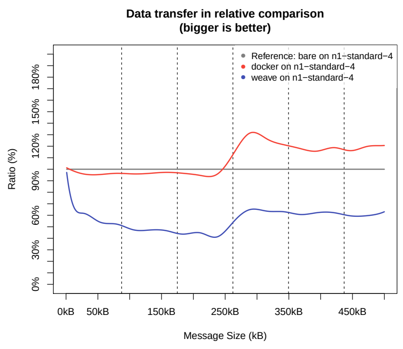
(b) AWS m3.xlarge (4 core)



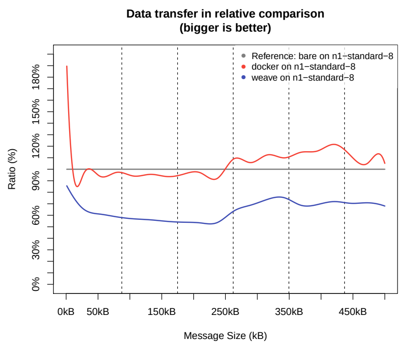
(c) AWS m3.2xlarge (8 core)



(d) GCE n1-standard-2 (2 core)



(e) GCE n1-standard-4 (4 core)



(f) GCE n1-standard-8 (8 core)

Figure 6. Relative transfer rates and TCP standard receive window of 83780 bytes (dotted lines)
$<$ 100% means performance loss, 100% means no loss, $>$ 100% means performance improvement

beyond 500kB messages. But of course, it might be possible that there exist further non-continuous network behavior points like we have seen for the Dart implementation of the *ping-pong* system. Furthermore, we only covered one SDN solution so far. Our ongoing efforts concentrate on extending the benchmarking solution *ppbench* by additional overlay networks like *Flannel* [19] (often used for *Kubernetes*), *Calico* [20] (a pure layer 3 approach to virtual networking for highly scalable data centers) or the recently released built-in overlay network feature of *Docker 1.9 (libnetwork)* to harden and eliminate possible solution specific conclusions.

So far, all of our study results indicate that our collected data can be used to do a systematic virtual machine type selection for cloud deployed HTTP-based and REST-like systems of general applicability. This is especially true for microservice based systems being deployed on container clusters like *Mesos*, *Kubernetes* or *Docker Swarm*.

It seems to be a simple and cost effective strategy to operate container clusters with **most similar high core machine types** across different providers.

This strategy should cover all relevant and complex performance related implications of containers, container clusters and SDNs appropriately. So, container and SDN technologies must not be avoided in general due to performance apprehensions. Under special circumstances and deployed on high core machine types, containers can even show better performances! Furthermore, container and SDN technologies provide more flexibility and manageability in designing complex horizontally scalable distributed cloud systems. But they should be always used with the above mentioned performance implications in mind.

Taking all together, a similarity-based virtual machine type selection can be used easily to optimize network performance for container cluster based cloud computing. Finally, the reader might be pleased to hear, that the selected machine types for this contribution fulfilled the criterion to be the most similar machine types across AWS and GCE, but these selected machine types are far from the most expensive machine types provided by both providers. Thus, a similarity-based virtual machine type selection seems to be a cost effective strategy as well.

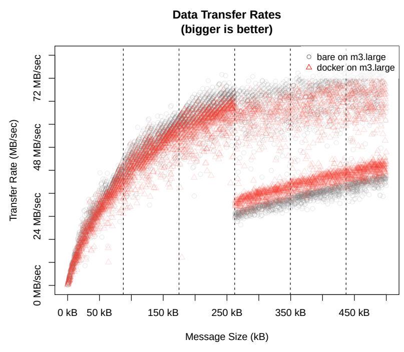
Acknowledgment

This study was funded by German Federal Ministry of Education and Research (Project Cloud TRANSIT, 03FH021PX4). We used research grants by courtesy of Amazon Web Services and Google Compute Engine. We also thank Bryan Boreham of Weaveworks for checking our *Weave* data for our initial conference paper [1]. We tried to realize his valuable insights and suggestions in this extended version. And finally, the authors thank Lübeck University (Institute of Telematics) and fat IT solution GmbH (Kiel) for their support of Cloud TRANSIT.

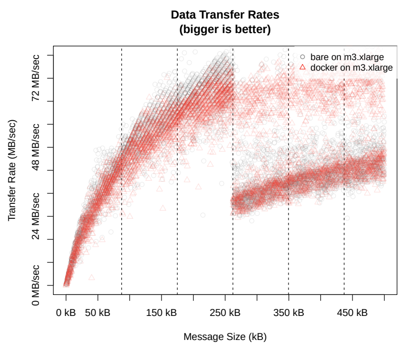
APPENDIX

TABLE IV. Similarity of AWS and GCE virtual machines types (sorted on both axis by descending amount of simultaneous executable threads).

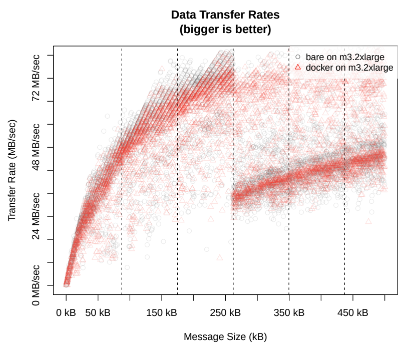
Similarities	n1-highmem-32	n1-standard-32	n1-highcpu-32	n1-highmem-16	n1-standard-16	n1-highcpu-16	n1-highmem-8	n1-standard-8	n1-highcpu-8	n1-highmem-4	n1-standard-4	n1-highcpu-4	n1-highmem-2	n1-standard-2	n1-highcpu-2	n1-standard-1	g1-small	f1-micro
m4.10xlarge	0.67	0.70	0.68	0.56	0.50	0.52	0.51	0.46	0.48	0.40	0.41	0.43	0.36	0.37	0.40	0.35	0.30	0.13
c4.8xlarge	0.66	0.66	0.70	0.59	0.65	0.54	0.55	0.47	0.50	0.41	0.43	0.46	0.37	0.39	0.43	0.38	0.32	0.14
c3.8xlarge	0.81	0.82	0.81	0.56	0.64	0.51	0.52	0.45	0.47	0.39	0.40	0.42	0.34	0.35	0.39	0.33	0.27	0.13
i2.4xlarge	0.61	0.57	0.60	0.84	0.77	0.80	0.66	0.60	0.64	0.52	0.54	0.59	0.46	0.48	0.53	0.47	0.40	0.22
r3.4xlarge	0.60	0.57	0.60	0.83	0.76	0.79	0.65	0.59	0.62	0.50	0.52	0.56	0.44	0.45	0.51	0.44	0.37	0.21
m4.4xlarge	0.61	0.58	0.61	0.84	0.77	0.80	0.66	0.60	0.64	0.51	0.54	0.58	0.45	0.47	0.52	0.46	0.39	0.21
c3.4xlarge	0.60	0.57	0.60	0.83	0.76	0.79	0.65	0.72	0.62	0.50	0.52	0.56	0.43	0.45	0.50	0.44	0.37	0.21
c4.4xlarge	0.59	0.55	0.58	0.82	0.75	0.79	0.64	0.72	0.62	0.53	0.56	0.60	0.49	0.51	0.56	0.50	0.44	0.25
i2.2xlarge	0.52	0.49	0.51	0.62	0.63	0.65	0.79	0.81	0.83	0.60	0.62	0.63	0.52	0.53	0.54	0.51	0.43	0.27
r3.2xlarge	0.52	0.49	0.51	0.62	0.63	0.65	0.79	0.81	0.83	0.60	0.62	0.63	0.52	0.53	0.54	0.51	0.43	0.27
m4.2xlarge	0.46	0.43	0.45	0.55	0.57	0.60	0.74	0.77	0.78	0.57	0.60	0.60	0.49	0.51	0.50	0.53	0.54	0.37
m3.2xlarge	0.52	0.49	0.50	0.61	0.63	0.66	0.79	0.96	0.83	0.60	0.63	0.63	0.52	0.54	0.54	0.52	0.44	0.27
c3.2xlarge	0.52	0.50	0.51	0.62	0.64	0.80	0.79	0.82	0.83	0.60	0.76	0.62	0.52	0.53	0.53	0.51	0.44	0.27
c4.2xlarge	0.53	0.51	0.52	0.62	0.64	0.80	0.78	0.81	0.82	0.58	0.74	0.60	0.50	0.52	0.52	0.50	0.42	0.26
i2.xlarge	0.45	0.42	0.43	0.54	0.59	0.59	0.58	0.78	0.62	0.81	0.82	0.80	0.60	0.60	0.58	0.57	0.48	0.32
r3.xlarge	0.46	0.42	0.43	0.54	0.59	0.60	0.58	0.79	0.63	0.82	0.83	0.81	0.62	0.62	0.60	0.58	0.50	0.33
m4.xlarge	0.47	0.44	0.45	0.56	0.61	0.61	0.60	0.66	0.64	0.83	0.85	0.82	0.63	0.63	0.60	0.60	0.51	0.34
m3.xlarge	0.47	0.43	0.45	0.56	0.58	0.74	0.60	0.63	0.64	0.80	0.96	0.82	0.59	0.61	0.61	0.58	0.49	0.31
c3.xlarge	0.46	0.42	0.43	0.54	0.60	0.59	0.58	0.65	0.77	0.82	0.83	0.79	0.61	0.74	0.57	0.56	0.48	0.32
c4.xlarge	0.49	0.46	0.47	0.57	0.62	0.62	0.61	0.67	0.79	0.81	0.83	0.80	0.59	0.74	0.57	0.57	0.48	0.32
r3.large	0.39	0.36	0.37	0.45	0.54	0.65	0.48	0.56	0.52	0.64	0.76	0.58	0.83	0.81	0.77	0.65	0.57	0.40
m3.large	0.41	0.38	0.39	0.48	0.54	0.53	0.51	0.58	0.69	0.63	0.64	0.61	0.84	0.98	0.80	0.67	0.58	0.39
c3.large	0.39	0.37	0.37	0.46	0.54	0.51	0.49	0.57	0.52	0.64	0.63	0.58	0.84	0.81	0.77	0.79	0.56	0.39
c4.large	0.40	0.37	0.37	0.46	0.55	0.51	0.50	0.58	0.53	0.65	0.63	0.59	0.81	0.79	0.75	0.78	0.56	0.40
t2.medium	0.43	0.40	0.40	0.49	0.56	0.52	0.53	0.59	0.55	0.66	0.62	0.59	0.74	0.70	0.68	0.69	0.48	0.31
m3.medium	0.31	0.30	0.30	0.34	0.42	0.39	0.36	0.43	0.40	0.50	0.47	0.44	0.57	0.55	0.51	0.86	0.77	0.57
t2.small	0.38	0.36	0.37	0.43	0.50	0.47	0.45	0.52	0.49	0.61	0.58	0.54	0.66	0.63	0.74	0.78	0.87	0.52
t2.micro	0.31	0.30	0.29	0.34	0.37	0.36	0.37	0.40	0.38	0.47	0.46	0.45	0.53	0.52	0.50	0.68	0.63	0.46



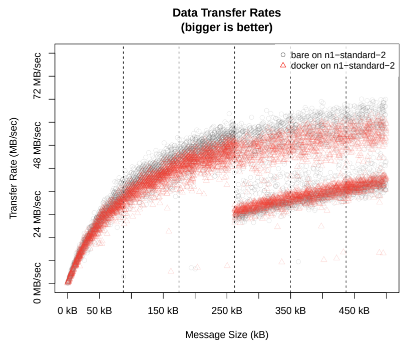
(a) AWS m3.large (2 core)



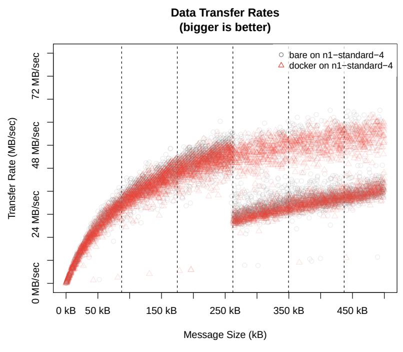
(b) AWS m3.xlarge (4 core)



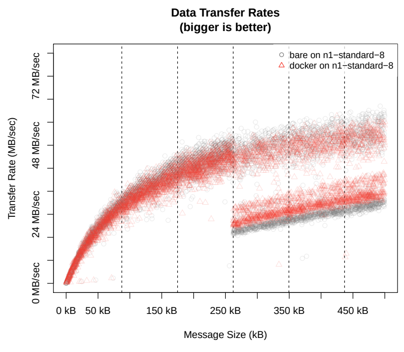
(c) AWS m3.2xlarge (8 core)



(d) GCE n1-standard-2 (2 core)



(e) GCE n1-standard-4 (4 core)



(f) GCE n1-standard-8 (8 core)

Figure 7. Bare and *Docker* experiments and TCP standard receive window of 83780 bytes (dotted lines). *Weave* data is not plotted for readability reasons.

References

[1] N. Kratzke, "About microservices, containers and their underestimated impact on network performance," CLOUD COMPUTING 2015, 2015, pp. 165–169.

[2] Docker. Last access 9th Nov. 2015. [Online]. Available: https://docker.com

[3] Weave. Last access 9th Nov. 2015. [Online]. Available: https://github.com/zettio/weave

[4] S. Newman, Building Microservices. O'Reilly and Associates, 2015.

[5] M. Fowler and J. Lewis. Microservices. Last access 17th Jul. 2015. [Online]. Available: http://martinfowler.com/articles/microservices.html [retrieved: March, 2014]

[6] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," SIGOPS Oper. Syst. Rev., vol. 41, no. 3, Mar. 2007, pp. 275–287.

[7] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," IBM Research Division, Austin Research Laboratory, Tech. Rep., 2014.

[8] Mesos. Last access 10th Dec. 2015. [Online]. Available: https://mesos.apache.org

[9] Coreos. Last access 9th Nov. 2015. [Online]. Available: https://coreos.com

[10] Kubernetes. Last access 17th Jul. 2015. [Online]. Available: https://github.com/GoogleCloudPlatform/kubernetes

[11] N. Kratzke, "A lightweight virtualization cluster reference architecture derived from open source paas platforms," Open Journal of Mobile Computing and Cloud Computing (MCCC), vol. 1, no. 2, 2014, pp. 17–30.

[12] R. Peinl, F. Holzschuher, and F. Pfitzer, "Docker cluster management - state of the art and own solution," Journal of Grid Computing, 2015.

[13] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[14] N. Kratzke. Ping pong - a distributed http-based and rest-like ping-pong system for test and benchmarking purposes. Last access 9th Nov. 2015. [Online]. Available: https://github.com/nkratzke/pingpong

[15] ——, "Lightweight virtualization cluster - howto overcome cloud vendor lock-in," Journal of Computer and Communication (JCC), vol. 2, no. 12, oct 2014.

[16] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 295–308. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972488

[17] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," Communications Surveys Tutorials, IEEE, vol. 17, no. 1, Firstquarter 2015, pp. 27–51.

[18] Open vswitch. Last access 9th Nov. 2015. [Online]. Available: http://openvswitch.org

[19] Flannel. Last access 9th Nov. 2015. [Online]. Available: https://github.com/coreos/flannel

[20] Project calico. Last access 9th Nov. 2015. [Online]. Available: http://www.projectcalico.org/

[21] D. Mosberger and T. Jin, "Httperf - a tool for measuring web server performance," SIGMETRICS Perform. Eval. Rev., vol. 26, no. 3, Dec. 1998, pp. 31–37. [Online]. Available: http://doi.acm.org/10.1145/306225.306235

[22] G. Memik, W. H. Mangione-Smith, and W. Hu, "Netbench: A benchmarking suite for network processors," in Proceedings of the 2001 IEEE/ACM International Conference on Computer-aided Design, ser. ICCAD '01. Piscataway, NJ, USA: IEEE Press, 2001, pp. 39–42.

[23] J. Verdú, J. Garcí, M. Nemirovsky, and M. Valero, "Architectural impact of stateful networking applications," in Proceedings of the 2005 ACM Symposium on Architecture for Networking and Communications Systems, ser. ANCS '05. New York, NY, USA: ACM, 2005, pp. 11–18.

[24] K. Velásquez and E. Gamess, "A Comparative Analysis of Network Benchmarking Tools," Proceedings of the World Congress on Engineering and Computer Science 2009 Vol I, Oct. 2009. [Online]. Available: http://www.iaeng.org/publication/WCECS2009/WCECS2009_pp299-305.pdf

[25] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. Wright, "Performance analysis of high performance computing applications on the amazon web services cloud," in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, Nov 2010, pp. 159–168.

[26] G. Wang and T. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in INFOCOM, 2010 Proceedings IEEE, March 2010, pp. 1–9.

[27] D. Jayasinghe, S. Malkowski, J. LI, Q. Wang, Z. Wang, and C. Pu, "Variations in performance and scalability: An experimental study in iaas clouds using multi-tier workloads," Services Computing, IEEE Transactions on, vol. 7, no. 2, April 2014, pp. 293–306.

[28] Berkley Lab, "iPerf - The network bandwidth measurement tool," https://iperf.fr, 2015.

[29] Sun Microsystems, "uperf - A network performance tool," http://www.uperf.org, 2012.

[30] netperf.org, "The Public Netperf Homepage," http://www.netperf.org, 2012.

[31] N. Kratzke and P.-C. Quint, "About automatic benchmarking of iaas cloud service providers for a world of container clusters," Journal of Cloud Computing Research, vol. 1, no. 1, 2015, pp. 16–34. [Online]. Available: http://jccr.uscip.us/PublishedIssues.aspx

[32] HP Labs, "httperf - A tool for measuring web server performance," http://www.hpl.hp.com/research/linux/httperf/, 2008.

[33] Apache Software Foundation, "ab - Apache HTTP server benchmarking tool," http://httpd.apache.org/docs/2.2/programs/ab.html, 2015.

[34] D. Ergu, G. Kou, Y. Peng, Y. Shi, and Y. Shi, "The analytic hierarchy process: Task scheduling and resource allocation in cloud computing environment," J. Supercomput., vol. 64, no. 3, Jun. 2013, pp. 835–848. [Online]. Available: http://dx.doi.org/10.1007/s11227-011-0625-1

[35] S. K. Garg, S. Versteeg, and R. Buyya, "Smicloud: A framework for comparing and ranking cloud services," in Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on. IEEE, 2011, pp. 210–218.

[36] C.-K. Ke, Z.-H. Lin, M.-Y. Wu, and S.-F. Chang, "An optimal selection approach for a multi-tenancy service based on a sla utility," in Computer, Consumer and Control (IS3C), 2012 International Symposium on. IEEE, 2012, pp. 410–413.

[37] A. Afshari, M. Mojahed, and R. M. Yusuff, "Simple additive weighting approach to personnel selection problem," International Journal of Innovation, Management and Technology, vol. 1, no. 5, 2010, pp. 511–515.

[38] C. Quinton, N. Haderer, R. Rouvoy, and L. Duchien, "Towards multi-cloud configurations using feature models and ontologies," in Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds. ACM, 2013, pp. 21–26.

[39] C.-W. Chang, P. Liu, and J.-J. Wu, "Probability-based cloud storage providers selection algorithms with maximum availability," in Parallel Processing (ICPP), 2012 41st International Conference on. IEEE, 2012, pp. 199–208.

[40] J. Siegel and J. Perdue, "Cloud services measures for global use: the service measurement index (smi)," in SRII Global Conference (SRII), 2012 Annual. IEEE, 2012, pp. 411–415.

[41] R. Karim, C. Ding, and A. Miri, "An end-to-end qos mapping approach for cloud service selection," in Services (SERVICES), 2013 IEEE Ninth World Congress on. IEEE, 2013, pp. 341–348.

[42] J. Yang, W. Lin, and W. Dou, "An adaptive service selection method for cross-cloud service composition," Concurrency and Computation: Practice and Experience, vol. 25, no. 18, 2013, pp. 2435–2454.

[43] C. Esposito, M. Ficco, F. Palmieri, and A. Castiglione, "Smart cloud storage service selection based on fuzzy logic, theory of evidence and game theory," Computers, IEEE Transactions on, vol. PP, no. 99, 2015, pp. 1–1.

[44] J. Stone, "Tachyon parallel / multiprocessor ray tracing system," http://jedi.ks.uiuc.edu/~johns/raytracer, 1995, accessed May 20, 2015.

[45] J. D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, Dec. 1995, pp. 19–25.

[46] W. D. Norcott and D. Capps, "Iozone filesystem benchmark," www.iozone.org, vol. 55, 2003.

[47] D. Lin, "An information-theoretic definition of similarity," in Proceedings of the Fifteenth International Conference on Machine Learning, ser. ICML '98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 296–304. [Online]. Available: http://dl.acm.org/citation.cfm?id=645527.657297

[48] Dart. Last access 9th Nov. 2015. [Online]. Available: https://www.dartlang.org/

[49] Apachebench. Last access 9th Nov. 2015. [Online]. Available: http://httpd.apache.org/docs/2.2/programs/ab.html

[50] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2014. [Online]. Available: http://www.R-project.org/

[51] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," RFC 1323 (Proposed Standard), Internet Engineering Task Force, May 1992, obsoleted by RFC 7323. [Online]. Available: http://www.ietf.org/rfc/rfc1323.txt

[52] D. Borman, B. Braden, V. Jacobson, and R. Scheffenegger, "TCP Extensions for High Performance," RFC 7323 (Proposed Standard), Internet Engineering Task Force, Sep. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7323.txt