# About Automatic Benchmarking of IaaS Cloud Service Providers for a World of Container Clusters

**ARTICLE** · JUNE 2015

READS
218

**2 AUTHORS:**

Nane Kratzke
Luebeck University of Applied Sciences
**30** PUBLICATIONS **38** CITATIONS
SEE PROFILE

Peter-Christian Quint
Luebeck University of Applied Sciences
**8** PUBLICATIONS **0** CITATIONS
SEE PROFILE

Columbia International Publishing
Journal of Cloud Computing Research
(2015) Vol. 1 No. 1 pp. 16-34

Research Article

# About Automatic Benchmarking of IaaS Cloud Service Providers for a World of Container Clusters

Nane Kratzke[1], Peter-Christian Quint[1*]

Received 9 April 2015; Published online 13 June 2015



## Abstract

Cloud service selection can be a complex and challenging task for a cloud engineer. Most current approaches try to identify a best cloud service provider by evaluating several relevant criteria like prices, processing, memory, disk, network performance, quality of service and so on. Nevertheless, the decision making problem involves so many variables, that it is hard to model it appropriately. We present an approach that is not about selecting a best cloud service provider. It is about selecting most similar resources provided by different cloud service providers. This fits much better practical needs of cloud service engineers. Especially, if container clusters are involved. EasyCompare, an automated benchmarking tool suite to compare cloud service providers, is able to benchmark and compare virtual machine types of different cloud service providers using an Euclidian distance measure. It turned out, that only 1% of theoretical possible machine pairs have to be considered in practice. These relevant machine types can be identified by systematic benchmark runs in less than three hours. We present some expectable but also astonishing evaluation results of EasyCompare used to evaluate two major and representative public cloud service providers: Amazon Web Services and Google Compute Engine.

*Keywords:* Cloud service selection; IaaS; Virtual machine; Performance; Cloud computing; Similarity measure; Performance vector; Benchmark; Container cluster

## 1. Introduction

Cloud service selection can be a complex and challenging task for a cloud engineer. According to Sun et al. [59], there exist several research directions like analytic hierarchy processes [16], utility function based methodologies [23], outranking approaches [30], simple additive weighting approaches[1], cloud feature models [48], dynamic programming approaches [9], cloud service indexing [54], ranking approaches [29], Markov decision process [63] or even astonishing combinations of Fuzzy logic, evidence theory and game theory [17] to support this task. But all of theses mentioned approaches seem to have the tendency to end in complex mathematical models. Due to this mathematical complexity these approaches seem to overstrain cloud engineers acceptance. These approaches are often exaggerated or not well accepted by practioners. In practice, cloud service selection is most of the time not realized very systematically. According to our experience, cloud service selection is more

*Corresponding email: nane.kratzke@fh-luebeck.de
1 Lübeck University of Applied Sciences, Department of Electrical Engineering and Computer Science, Center of Excellence CoSA

an evolutionary process. So often, the first cloud service provider is not selected systematically, it is just a "mean available".

Therefore, our guiding question in this paper is an engineering one. It is not about selecting virtual machines from a best cloud service provider, but it is about selecting the most similar cloud virtual machines provided by different cloud service providers. Container clusters like Mesos [6], CoreOS [12] or Kubernetes [24] are getting more and more attention in cloud computing due to the fact that they are providing a horizontally scalable concept of one logical virtual machine, that can be deployed on up to thousands of "physical" nodes. These "physical" nodes can be hosted by different public providers and in private clouds or on bare-metal machines as well. Horizontally scalable microservice approaches – widely used by companies like Facebook, Twitter, Netflix, and so on – rely heavily on this kind of cluster solutions [43]. This is done to handle regional workloads, to provide failover and overflow capacities and also to avoid cloud vendor lock-in [32], [33]. All provided machines of such kind of cluster should show similar performance characteristics to provide fine-grained resource allocation capabilities [27]. At this point, one obvious question arises: How to identify similar virtual machine types from different providers? None of the above mentioned approaches will answer this question. All of them are trying to identify a best (in terms of performance or cost-effectiveness) resource. These approaches are not helpful to identify most similar resources provided by different providers. A major problem is, that no two cloud service providers provide exactly the same virtual machine types.

To answer these questions, we propose an automated benchmarking system called EasyCompare. The core idea is to describe virtual machine type performances by a performance vector. A similarity value between 0 (not similar) and 1 (very similar) can be calculated using these vectors. The presented approach can be used to compare regional differences of providers, to compare machine variants of the same or different cloud service providers, or to define machine types for a private cloud to align them to performance of machines types of public cloud service providers, and so on.

The remainder of this paper is structured as follows. A proposed benchmarking architecture to collect benchmark data in the cloud is explained in Section 2. Section 3 analyzes related work and existing benchmarking solutions for cloud computing to derive some appropriate benchmarks to measure relevant performance aspects of cloud provided virtual machines. Section 4 presents some data, which has been collected to compare two representative public IaaS Cloud Service Providers (Amazon Web Services, AWS and Google Compute Engine, GCE). By analyzing this data, we will see that different cloud service providers provide a variety of comparable and non comparable machine types, making it complicated (but not impossible) to compare virtual machine types of different cloud service providers. Section 5 applies these insights to derive a similarity measure, making it possible to compare two arbitrary virtual machine types by a normalized Euclidian distance measure. The proposed Euclidian distance measure is evaluated on the collected data. We conclude our findings in Section 6 and show that our approach contributes to do a pragmatic cloud service selection only on relevant subsets of cloud offerings.

# 2. Architecture of Automated Benchmarking

The general architecture of EasyCompare is shown in Figure 1. It is a Ruby command line tool which has been developed by authors of this contribution to collect benchmark data of various cloud service providers. It has an extendable architecture to support arbitrary cloud service providers by plugins. Provider internals are handled by provider plugins for specific cloud service providers. The architecture is easily extendable for other providers (see Figure 1). At its current development state plugins for two prominent and representative public cloud service providers are provided: AWS and GCE. An OpenStack (an private cloud infrastructure) plugin is under development. Nevertheless, EasyCompare is designed to work with every public or private IaaS cloud infrastructure. It provides a data collection module to collect benchmark data from one or more IaaS cloud infrastructures and it provides an analytics module to analyze collected benchmark data.

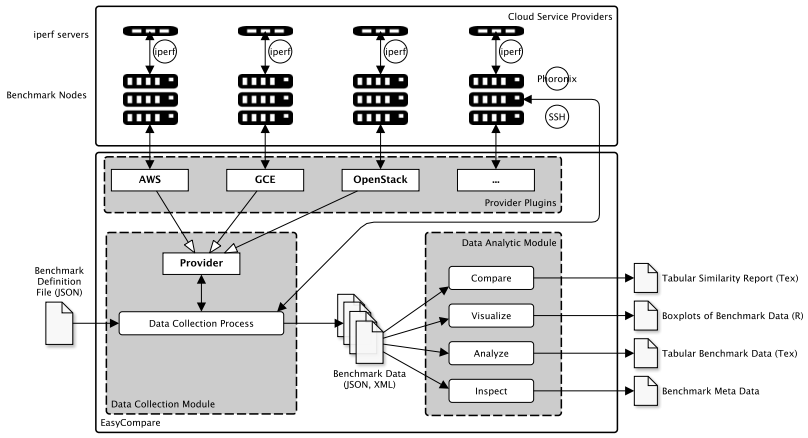


**Fig. 1**: Architecture of EasyCompare

## 2.1 *EasyCompare's Data Collection Module*

The data collection process presented in Figure 2 is executed by the data collection module of EasyCompare. The data collection can be configured by a benchmark definition file (JSON format). This file is used to define the provider, the datacenter (zone, region, etc.) and virtual machine types to be benchmarked. The file also contains the iperf configuration [37] for network benchmarking. One benchmark definition file can contain a set of arbitrary cloud service providers which are benchmarked in parallel (and automatic). For each provider the following benchmark process is executed in parallel.

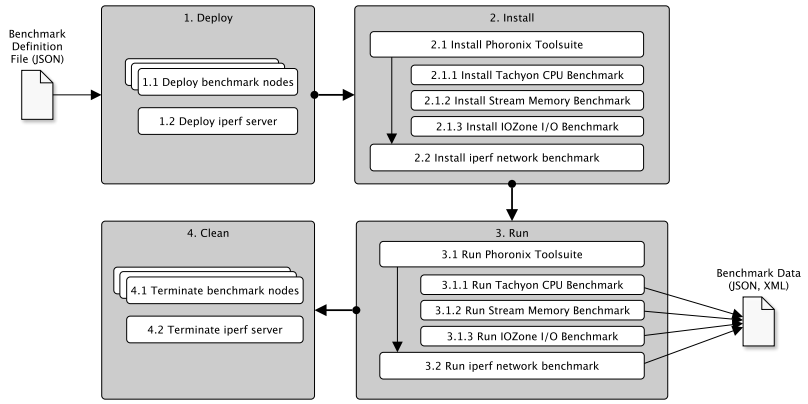


**Fig. 2**: Data Collection Process

In a **deploy step** all to be benchmarked virtual machines are deployed as well as the iperf server according to the definitions provided in JSON file. A corresponding benchmark node is deployed for each to be benchmarked virtual machine type. This node is used to collect relevant benchmark data of the virtual machine type. The following install and run steps are performed per benchmark node.

The **install step** waits for all deployed benchmark nodes to be booted and installs the necessary benchmarking software. This is realized via SSH remote shell commands executed via a Node interface. Therefore on all nodes an automated benchmark suite (Phoronix [46]) is installed and configured to operate non interactively and fully

automatic. Furthermore, Phoronix is used to install and configure all used benchmarks. The used benchmarks are introduced in Section 3. To do network benchmarking, iperf is installed as well.
The **run step** waits for all benchmarks to be installed and tells the Phoronix benchmark suite to execute them sequentially on all benchmark nodes. Phoronix generates XML files with benchmark data. These XML files are returned and saved to be analyzed and evaluated by the data analytic module (Section 2.2). It is up to the responsible benchmarking analyst, to select a virtual machine type for the iperf server. It must be assured, that networking bottlenecks are on the benchmark node's side and not on the iperf server's side (normally a virtual machine type with maximum network performance of a provider should be selected to play the role of the iperf server). The measured iperf benchmark data is returned and saved to be analyzed and evaluated by the data analytic module.
The **clean step** simply shuts down all deployed benchmark and iperf server nodes to avoid costs. So a complete benchmarking run against cloud service providers can be executed completely non-interactively.

### 2.2 *EasyCompare's Data Analytics Module*

The analytics module of EasyCompare provides several commands to inspect, analyze, visualize and compare collected benchmark data. All data provided to the following analytical commands can be filtered by providers, datacenters or virtual machine types of analytical interest.
The **inspect** command is used to identify high level meta data about benchmark data which has been recorded by the collect module. It reports for which cloud service providers, datacenters and machine types benchmark data are present. This meta data is primarily useful for analysts to filter existing benchmark data by the following analytic commands.
The **analyze** command can be used to generate detailed tabular reports of benchmark data, which has been recorded by the collect module. It is able to generate tabular reports like shown exemplary in Table 4.
The **visualize** command can be used to generate detailed visualized formats of benchmark data, which has been recorded by the collect module. The values are presented as box plots to enable estimation about variances and skewness of measured distributions. It is able to generate scripts, which can be processed by the statistical computing framework R [57] to generate box plot visualizations like shown exemplary in Figure 3.
The **compare** command can be used to compare benchmark data recorded by the collect module. Data presentation is done by tabular formats to compare efficiently a huge amount of machine type pairs from various providers. With this command it is also possible to generate Tex snippets, which can be processed by the Tex typesetting system [31] to generate similarity tables like shown exemplary in Table 1. The similarity of machine types is calculated by a normalized Euclidian distance measure. Details about the normalized Euclidian distance measure are explained in Section 5.

## 3. Benchmark Design

Benchmarking can be tricky in its details [60]. Especially if benchmarks are used to determine similarity across several varying instance types. Therefore, several publications and approaches which are often used to evaluate performance aspects of cloud provided virtual machines have been analyzed [7], [10], [19], [25], [28], [34], [41], [56], [61].

**Domain or application based benchmarks:** **CloudStone** is a toolkit for characterizing the workload of a typical Web 2.0 social networking application. CloudStone includes Faban [18], a workload framework designed for non static and distributed web applications. CloudStone is capable to use the measurement results for calculating a metric that quantifies the dollars per user per month [56]. **HiBench** is a benchmark suite for the open-source MapReduce implementation Apache Hadoop [3]. The suite contains real-world applications for web-search indexing like *Nutch Indexing* and *PageRank* [64], Hadoop distributed file system [4], [28] and benchmarks for large-scale machine learning like *Bayesian Classification* and *K-means Clustering*. HiBench contains

further benchmarks (like *Sort*, *WordCount* and *TeraSort* providing good performance indicators for MapReduce [14]) [28]. **CloudSuite** is a benchmark suite designed for scale-out workload experiments [20],[45]. The suite includes tasks that were identified as some of the more common tasks which are handled using cloud computing: *Data Analytics* using Apache Hadoop MapReduce framework [3] and the Apache Mahout library [5]. It is designed for machine learning analysis on large-scale datasets [45], *Data Caching* using Memchached [21], *Data Serving* using Cassandra [2] and Yahoo! Cloud Serving Benchmark [11], *Media Streaming* using the Darwin Streaming Server with workload by Faban [18], *graph based machine learning* using GraphLab [13], *Web Serving* using CloudStone [56], *Web Search* using (comparable to HiBench and CloudSuite) the Nutch search engine to benchmark the indexing process [45].

**Synthetic benchmarks:** **CloudCmp** is a very (and maybe the most) influencing benchmark in this domain and a framework designed for customers to choose their cloud provider. The framework compares performance and price to calculate the best cloud provider services by the needed technical requirements. It relies on benchmark suites like SPECjvm2008 [53], iperf [37], SciMark FFT, Crypto.AES, Crypto.RSA and further benchmarks. CloudCmp can calculate factors like cost per task [34]. **PerfKit Benchmarker** *"contains set of benchmarks to measure and compare cloud offerings. The benchmarks are not tuned (i.e. use defaults) to reflect what most users will see"* [25]. PerfKit contains a lot of benchmarks which makes this approach very run time intensive. And a lot of benchmarks seem to generate redundant performance data. Some of the used benchmarks by PerfKit are bonnie [8], CoreMark [15], iperf and netperf, Spec CPU 2006 [26], Unixbench [55], HPCC [36]. **ALIBI** [10] is a prototype of a resource monitoring system. It is designed to analyze nested virtualization overheads by using synthetic benchmarks for CPU/Memory and I/0 workloads. For compute- and memory-bound workloads SPEC CINT2006 [39] and for network I/O netperf [51] are used.

**Conclusion for our benchmark design:** We decided not to follow application based benchmarking approaches like CloudSuite or CloudStone. These approaches try to cover specific applications like Hadoop or domains like web serving or data streaming. We do not assume the usage IaaS cloud services for a specific domain or application type. Therefore, synthetic-like benchmarks to measure and compare virtual machine types in general terms of processing, memory, disk and networking performance are preferred to cover relevant performance attributes identified – for instance – by approaches like **CloudGenius** [41]. Therefore, our approach follows more the benchmark design philosophy of PerfKit or ALIBI without copying it exactly. Other than PerfKit or ALIBI, the amounts of benchmarks are reduced to generate a set of performance data without redundancies and in less time. But like PerfKit, benchmarks are configured by using their default values to reflect what most users will see.
Benchmark runs should have short run times for public cloud computing. Mainly, to reduce data collection costs. On the other hand, the provided data has to be statistically significant. Small sample sizes and significance must be no conflict but handled with care (see Schmid and Huber [52]). The benchmarks should be simply deployable and executable on virtual machines. And the returned data should be simply analyzable. That is, why data formats for benchmark reports like JSON or XML are used by EasyCompare. Automated benchmark toolsets like Phoronix [46] are simply deployable. Furthermore, these tools even check deviations while measuring benchmark results. If necessary (due to intolerable deviations of already collected data), additional benchmark runs are executed to provide statistically significant samples.

- The **Taychon** benchmark is used to measure the **processing performance** of a virtual machine type [58]. The Tachyon benchmark scales well with increasing cores and is faster than other benchmarks used by CloudCmp or PerfKit (SPECjvm2008 [53], SciMark FFT [47]). Finally, Tachyon has been selected to be part of the SPEC MPI2007 [42] benchmark suite. Nevertheless, like CloudCmp, Tachyon covers floating point operations. Tachyon benchmark is nothing more than rendering several standardized raytracing scenes with a same named raytracing library. Furthermore, the Tachyon benchmark is relatively independent from memory sizes and measures processing performance only.

- The **Stream** benchmark is used to measure the **memory performance** [38]. This is basically the same approach like CloudCmp. Stream is a simple, synthetic benchmark designed to measure sustainable memory bandwidth (in MB/s) for four simple vector operations (Copy, Scale, Add and Triad). To reduce benchmarking times we only applied the Triad operation. The triad operation has the most practical relevance according to Raman [50]. The other operations Copy, Scale, Add are not used.
- The **IOZone benchmark** [44] is used to measure read and write **disk performance**. The IOZone benchmark produces reliable results in shorter time than a DFSIO benchmark (often used by cloud benchmarks). To reduce benchmarking duration, only the read and write modes of 512 MB files with a record size of 1 MB are executed. Other modes (covering smaller and bigger file sizes and different record sizes) of the IOZone benchmark are not executed. Most of the times additional modes generate no fundamental new insights in the I/O performance of a system.
- Like CloudCmp and PerfKit do, **iperf** is used to measure **network transfer rates** of intra cloud data transfers. And like done by [10], data transfer rates are measured on the sender side to minimize communication counter part influences. iperf is used to send 30 times as many data as in 10 seconds possible to measure maximum transfer rates. Unlike CloudCmp, no wide-area delivery transfers are measured because there are to much influences, which can not be accounted reliably to a cloud service provider.

The above described benchmark suite is designed to be run in less than two hours even on small virtual machine types.

# 4. Performance Evaluation of two Cloud Service Providers

In Section 3 introduced benchmarks have been collected by EasyCompare (Section 2). Benchmark data has been measured on various machine types provided by AWS and GCE (see Appendix, Table 4). Benchmarking has been performed in several datacenters[1] of both providers and at different days[2] to measure a realistic distribution and not a unique distribution of performance characteristics. Median values, amount of samples $n$ and quartiles (in box plots) are presented according to recommendations, if nothing can be assumed on underlying probability distributions [52]. The variance and skewness of distributions can be estimated for each provider by provided box plots.

## 4.1 *Processing Performance*

Figure 3 shows that processing performance of a virtual machine is mainly influenced by the amount of available simultaneous executable threads. A $n$-thread machine is providing almost the same processing performance on GCE as on AWS. Even processing optimized machines do not show a substantial better performance than standard machines. So, provider labels like "computing" or "high-cpu" for machine types are mainly marketing labels. Both providers have low cost instances which are single thread processors and which show significant lower processing performance than all other provided machine types ('`m3.medium`' for AWS and '`n1-standard-1`' for GCE). While all machine types show almost no variance for processing performance, this is not true for the low-cost GCE machine type '`n1-standard-1`'. This machine type shows a significant performance variance (see Figure 3). This might be an indication that this machine type is a bit overbooked in the GCE infrastructure. Taking all together, machine types can be well compared by their available amount of simultaneous executable threads.

[1] AWS: eu-central-1, us-east-1; GCE: europe-west1-d, europe-west1-c, us-central1-f, asia-east1-a
[2] 25th Feb., 26th Feb., 27th Feb., 1st Mar., and 2nd Mar. 2015

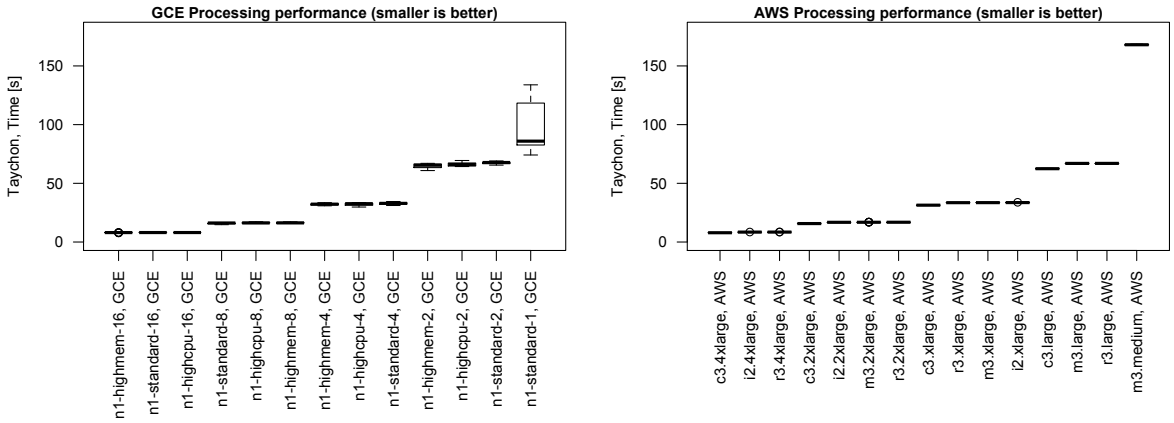


**Fig. 3**: Measured GCE and AWS processing performances

## 4.2 *Memory Performance*

More variances are likely for the memory performance of virtual machines. The reader can see this effect in Figure 4 by increasing box sizes. In general, memory performance is decreasing with processor performance (so the memory performance is aligned to the processing performance by public cloud providers, which is more than reasonable). AWS provides for high $n$-thread machines better memory performances than GCE. Similar to marketing labels for processing machines, there seem to exist marketing labels for memory optimized machines. Machines being labeled as RAM-optimized (AWS, `r3.*`) or "highmem" (GCE) seem not to show better transfer rates than standard machines (of course these machines provide substantial more memory, but the memory is not faster or shows smaller variances in data transfer rates, check Table 4).

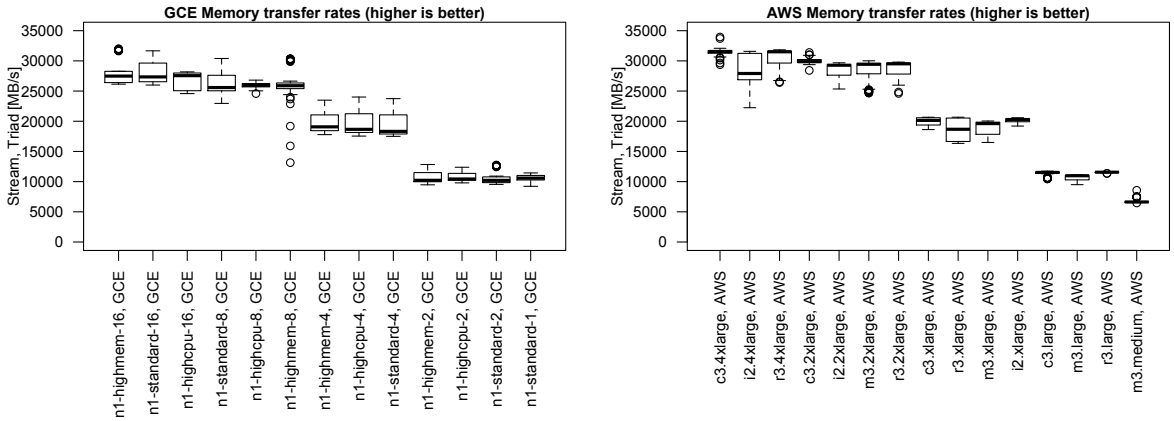


**Fig. 4**: Measured GCE and AWS memory transfer performances

### 4.3 *Disk Performance (read and write)*

I/O performances have the tendency to show even more variances than memory performances. This is the nature of things and not very astonishing. This can be seen for GCE and AWS as well. AWS seems to provide a slightly better read and write performance than GCE (see Figure 5). I/O performances of both providers decrease slightly with processing performances of machines. AWS shows better performances but also greater variances in data transfer rates. This is especially true for write performances of non low-cost machines. On the other hand, it must be stated that median write performance values of AWS are always on the upper side. So most of the write operations are fast and only some are slow. In general, AWS seems to be the better provider for storage intensive applications compared with GCE (if no low-cost machines are used). GCE shows more balanced I/O performance across all machine types but on a lower level.
Machines which are labeled as storage optimized (AWS: all i2 instance types are optimized for storage intensive tasks) seem not to show better transfer rates than standard machines. Of course these machines can be configured to provide substantial more disk space than standard machines. But the transfer rates are not faster or show smaller variances (check Table 4). Especially the '`i2.4xlarge`' instance type provides astonishing small read performances. At this point we have to say, that AWS and GCE provide options to optimize I/O performance by using SSDs (GCE and AWS) and assured I/O operations per second (AWS). We have only measured the default configuration of the mentioned machine types.
So, there exist a plenty of variations between different providers regarding disk performances which makes it complicated to compare the various machine types regarding their IO performance.

### 4.4 *Network Performance*

Regarding network performances of provided machines, a complete different philosophy of AWS and GCE can be seen. The AWS philosophy seems to provide a well defined and leveled network performance which is decreasing with decreasing processing performance of virtual machine type. On these well defined levels of network performance only minor variances in data transfer rates occur (see Figure 6).
On the other hand, the reader can see a complete different GCE network design philosophy (see Figure 6). Comparable to IO performance, GCE seems to provide a well-balanced network performance across all machine types. This transfer performance seems not even aligned to the processing performance of the machine types. In other words, GCE seems to try providing full network power for all machine types. This is bought in by substantial bigger variances of data transfer rates. Some GCE machine types '`n1-highmem-16`' and '`n1-highmem-8`' seem to show significantly higher data transfer rates. Nevertheless, their boxes are overlapping with a lot of other machines and so it is not clear whether this is intended by GCE or only a effect which has been measured coincidentally.
So, measured network performances of GCE and AWS show completely different design philosophies for the network design of a public cloud, making it even more complicated to compare the various machine types regarding their network performance.

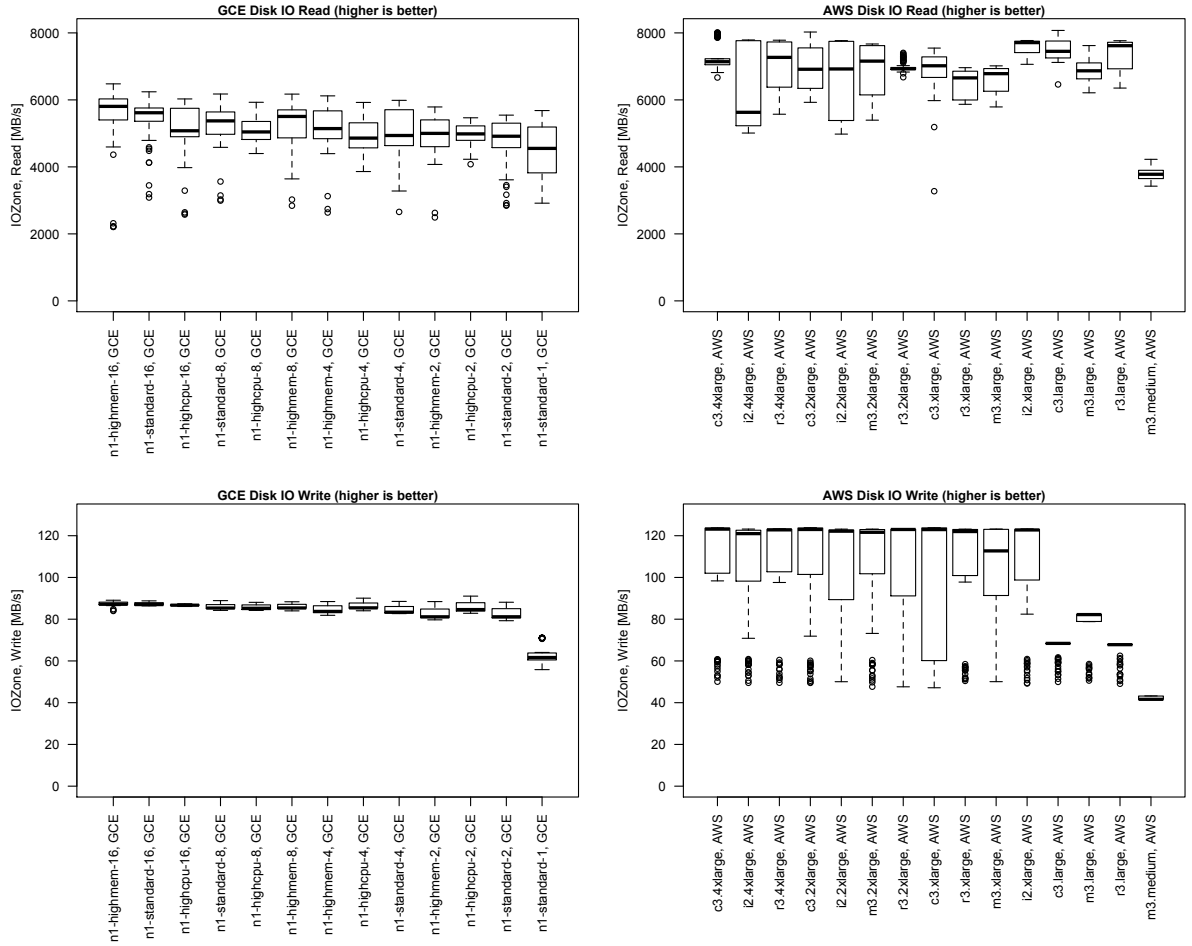


**Fig. 5**: Measured GCE and AWS disk IO performances

# 5. Describing and Comparing Virtual Machine Types

This contribution uses the following performance describing vector **i** to express virtual machine type performances provided by different cloud service providers (see Section 4).

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

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

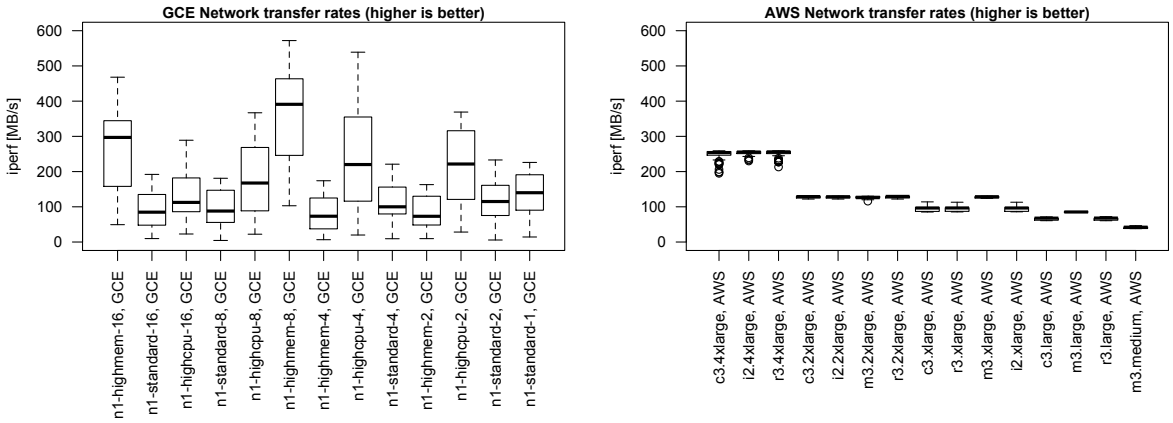


**Fig. 6**: Measured GCE and AWS network performances

similarity measure has to be chosen.

### 5.1 *Choosing an Appropriate Similarity Measure*

According to Lin, many similarity measures have been proposed [35]. McGill surveyed and compared over 60 similarity measures alone [40]. A good similarity measure has to fulfill three similarity intuitions (see Lin [35]).

- **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."*

In our case, machine performances are described as numerical feature vectors. This reduces the set of possible similarity measures to some promising approaches. Comparing two numerical feature vectors **i** and **j** is often done in information retrieval by the **cosine similarity** [22] or the **dice similarity** [35]

$$s_{\text{cosine}}(\mathbf{i},\mathbf{j}) = \frac{\sum_{n=1}^{m}(i_n \cdot j_n)}{\sqrt{\sum_{n=1}^{m}(i_n)^2} \cdot \sqrt{\sum_{n=1}^{m}(j_n)^2}} \qquad s_{\text{dice}}(\mathbf{i},\mathbf{j}) = \frac{2\sum_{n=1}^{m} i_n j_n}{\sum_{n=1}^{m} i_n^2 + \sum_{n=1}^{m} j_n^2} \tag{2}$$

or via distance measures like **euclidian distance** [49, 62] or derived **euclidian distance metric** [35].

$$d_{\text{euclidian}}(\mathbf{i},\mathbf{j}) = \sqrt{\sum_{n=1}^{m}(i_n - j_n)^2} \qquad m_{\text{euclidian}(\mathbf{i},\mathbf{j})} = \frac{1}{1 + d_{\text{euclidian}}(\mathbf{i},\mathbf{j})} \tag{3}$$

Sadly, it turned out, that all measures do not fulfill the second similarity intuition postulated by Lin [35] or had other shortcomings. The **Cosine similarity** tends to rate a lot of non similar machines to be similar. The **Dice similarity** tends to rate a lot of non-similar machines to be similar (although showing somewhat better results than the Cosine similarity). The **Euclidian distance** is able to identify identical machines as totally similar and showed reasonable results for second intuition similarities, but not in a normalized way. So a Euclidian similarity value of 0 means total similarity but what does a Euclidian similarity value of 500 means? The **Euclidian**

**distance metric** provides a normalized value but tends to behave like a boolean identical operator (it only was able to identify identical machines but rated all other machines as totally non similar, this has to do with the fact that the distance metric takes absolute performance differences into account and no performance relations). Nevertheless, measures comparing positions of a vector (Euclidian distances) showed more intuitive results compared to measures comparing vector directions (like Cosine similarity). Therefore, this contribution proposes a normalized Euclidian distance metric based on relative performance differences. The following variant of the Euclidian distance metric measures the relative performance relations of performance components $(i_1, i_2, \ldots, i_m)$ and $(j_1, j_2, \ldots, j_m)$ and normalizes the result $s(\mathbf{i}, \mathbf{j})$ to a similarity value between $0.0$ (**i** and **j** are not similar at all) and $1.0$ (**i** and **j** are completely similar).

$$\begin{aligned}
\forall \mathbf{i}, \mathbf{j} \quad & s(\mathbf{i}, \mathbf{j}) = 1 - \frac{1}{m} \sum_{n=1}^{m} \left(1 - \frac{\min(i_n, j_n)}{\max(i_n, j_n)}\right)^2 \\
\forall \mathbf{i}, \mathbf{j} \quad & s(\mathbf{i}, \mathbf{j}) = s(\mathbf{j}, \mathbf{i}) \\
\forall \mathbf{i}, \mathbf{j} \quad & 0 \leq s(\mathbf{i}, \mathbf{j}) \leq 1 \\
\forall \mathbf{i} \quad & s(\mathbf{i}, \mathbf{i}) = 1
\end{aligned} \tag{4}$$

It produces similarity values which are in accordance with cloud engineer expectations and fulfills all relevant similarity intuitions postulated by Lin [35].
Lin's concept of similarity is designed as an information-theoretic definition and therefore has to consider missing data as well. Missing data can not happen in our case. We had only to consider commonality (intuition 1), if we would like to compare non complete numerical feature vectors. But that is not the case and so we do not have to consider Lin's intuition 1 for evaluation. But of course, we have to consider the second and third intuition and show how the proposed similarity measure is in accordance to these intuitions. We do this by showing that the proposed similarity measure fulfills the identical intuition (intuition 3) in Section 5.2 and the difference intuition (intuition 2) in Section 5.3 and Section 5.4.

### 5.2 *Evaluating the Distance Measure by Comparing Machine Types of the same Provider*

Table 1 shows exemplary the Euclidian similarity of various GCE machine types. On the main diagonal of Table 1 all similarity values are 1.0 and this must be true according to Lin's third identical intuition because on the main diagonal identical machine types are compared. And according to our expectation, similarities are decreasing with increasing processing, memory, disk, or networking differences of virtual machine types (which is according to Lin's second difference intuition). Furthermore, we can even identify some similarity clusters in Table 1 which are aligned to the amount of simultaneous executable threads. 16 thread machines show comparable similarity values, the same is true for 8, 4 and 2 thread machines. The same effect can be seen for AWS machine types as well (but due to page limitations, data is not presented).
The Euclidian distance measure works well with both providers. But we are going to double check that by comparing same machine types of different datacenters of the same provider and by comparing different machine types of different providers.

### 5.3 *Evaluation the Distance Measure by Comparing Provider Datacenters*

Machine types from different datacenters of the same provider should show similar characteristics. Nevertheless, it is not realistic to expect a perfect similarity value of 1.0 on the main diagonal. But the resulting similarity table should look very similar to the Table 1. And of course, slightly reduced similarity values are expectable due to statistical fluctuations. And this is exactly what can be seen exemplary in Table 2 for GCE. There still exist a clear (but not longer perfect) main diagonal. But almost all similarity values are decreasing and no exact similarities exist any more.

**Table 1**: Euclidian similarity values of GCE machine types (sorted on both axis by descending amount of simultaneous executable threads)

| Similarities | n1-highcpu-16 (GCE) | n1-highmem-16 (GCE) | n1-standard-16 (GCE) | n1-highcpu-8 (GCE) | n1-highmem-8 (GCE) | n1-standard-8 (GCE) | n1-highcpu-4 (GCE) | n1-highmem-4 (GCE) | n1-standard-4 (GCE) | n1-highcpu-2 (GCE) | n1-highmem-2 (GCE) | n1-standard-2 (GCE) | n1-standard-1 (GCE) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| n1-highcpu-16 (GCE) | 1.00 | 0.81 | 0.85 | 0.66 | 0.62 | 0.67 | 0.60 | 0.60 | 0.76 | 0.53 | 0.53 | 0.54 | 0.53 |
| n1-highmem-16 (GCE) | 0.81 | 1.00 | 0.79 | 0.65 | 0.68 | 0.61 | 0.60 | 0.54 | 0.56 | 0.54 | 0.47 | 0.49 | 0.49 |
| n1-standard-16 (GCE) | 0.85 | 0.79 | 1.00 | 0.64 | 0.60 | 0.68 | 0.57 | 0.62 | 0.62 | 0.51 | 0.55 | 0.54 | 0.52 |
| n1-highcpu-8 (GCE) | 0.66 | 0.65 | 0.64 | 1.00 | 0.83 | 0.82 | 0.67 | 0.62 | 0.64 | 0.58 | 0.54 | 0.70 | 0.55 |
| n1-highmem-8 (GCE) | 0.62 | 0.68 | 0.60 | 0.83 | 1.00 | 0.78 | 0.64 | 0.58 | 0.60 | 0.56 | 0.50 | 0.51 | 0.51 |
| n1-standard-8 (GCE) | 0.67 | 0.61 | 0.68 | 0.82 | 0.78 | 1.00 | 0.63 | 0.66 | 0.67 | 0.54 | 0.58 | 0.58 | 0.55 |
| n1-highcpu-4 (GCE) | 0.60 | 0.60 | 0.57 | 0.67 | 0.64 | 0.63 | 1.00 | 0.80 | 0.82 | 0.65 | 0.60 | 0.62 | 0.61 |
| n1-highmem-4 (GCE) | 0.60 | 0.54 | 0.62 | 0.62 | 0.58 | 0.66 | 0.80 | 1.00 | 0.84 | 0.59 | 0.65 | 0.62 | 0.59 |
| n1-standard-4 (GCE) | 0.76 | 0.56 | 0.62 | 0.64 | 0.60 | 0.67 | 0.82 | 0.84 | 1.00 | 0.61 | 0.64 | 0.65 | 0.62 |
| n1-highcpu-2 (GCE) | 0.53 | 0.54 | 0.51 | 0.58 | 0.56 | 0.54 | 0.65 | 0.59 | 0.61 | 1.00 | 0.80 | 0.83 | 0.68 |
| n1-highmem-2 (GCE) | 0.53 | 0.47 | 0.55 | 0.54 | 0.50 | 0.58 | 0.60 | 0.65 | 0.64 | 0.80 | 1.00 | 0.84 | 0.67 |
| n1-standard-2 (GCE) | 0.54 | 0.49 | 0.54 | 0.70 | 0.51 | 0.58 | 0.62 | 0.62 | 0.65 | 0.83 | 0.84 | 1.00 | 0.69 |
| n1-standard-1 (GCE) | 0.53 | 0.49 | 0.52 | 0.55 | 0.51 | 0.55 | 0.61 | 0.59 | 0.62 | 0.68 | 0.67 | 0.69 | 1.00 |

The same effects (but less distinctive) can be seen for AWS. The reader can see that the Euclidian distance measure works well with both providers and is even able to compare datacenter definition characteristics of providers. It fulfills Lin's second intuition (difference intuition) and works well close to Lin's third intuition (identical intuition).

**Table 2**: Euclidian similarity values of various GCE machine types hosted in `europe-west1-c` and `us-central1-f` (sorted on both axis by descending amount of simultaneous executable threads)

| Similarities | n1-highcpu-16 (GCE, us-central1-f) | n1-highmem-16 (GCE, us-central1-f) | n1-standard-16 (GCE, us-central1-f) | n1-highcpu-8 (GCE, us-central1-f) | n1-highmem-8 (GCE, us-central1-f) | n1-standard-8 (GCE, us-central1-f) | n1-highcpu-4 (GCE, us-central1-f) | n1-highmem-4 (GCE, us-central1-f) | n1-standard-4 (GCE, us-central1-f) | n1-highcpu-2 (GCE, us-central1-f) | n1-highmem-2 (GCE, us-central1-f) | n1-standard-2 (GCE, us-central1-f) | n1-standard-1 (GCE, us-central1-f) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| n1-highcpu-16 (GCE, europe-west1-c) | 0.98 | 0.86 | 0.79 | 0.65 | 0.66 | 0.62 | 0.61 | 0.54 | 0.73 | 0.54 | 0.48 | 0.52 | 0.51 |
| n1-highmem-16 (GCE, europe-west1-c) | 0.83 | 1.00 | 0.79 | 0.65 | 0.67 | 0.62 | 0.61 | 0.54 | 0.59 | 0.54 | 0.48 | 0.51 | 0.51 |
| n1-standard-16 (GCE, europe-west1-c) | 0.83 | 0.80 | 1.00 | 0.65 | 0.60 | 0.67 | 0.56 | 0.61 | 0.59 | 0.49 | 0.55 | 0.53 | 0.50 |
| n1-highcpu-8 (GCE, europe-west1-c) | 0.68 | 0.66 | 0.64 | 1.00 | 0.81 | 0.84 | 0.66 | 0.62 | 0.66 | 0.57 | 0.55 | 0.72 | 0.56 |
| n1-highmem-8 (GCE, europe-west1-c) | 0.62 | 0.65 | 0.59 | 0.80 | 1.00 | 0.78 | 0.65 | 0.57 | 0.61 | 0.56 | 0.50 | 0.53 | 0.52 |
| n1-standard-8 (GCE, europe-west1-c) | 0.67 | 0.65 | 0.66 | 0.85 | 0.79 | 0.99 | 0.63 | 0.64 | 0.66 | 0.55 | 0.57 | 0.58 | 0.55 |
| n1-highcpu-4 (GCE, europe-west1-c) | 0.61 | 0.59 | 0.59 | 0.66 | 0.60 | 0.65 | 0.98 | 0.81 | 0.86 | 0.63 | 0.63 | 0.66 | 0.62 |
| n1-highmem-4 (GCE, europe-west1-c) | 0.59 | 0.56 | 0.62 | 0.64 | 0.58 | 0.66 | 0.80 | 0.98 | 0.83 | 0.60 | 0.65 | 0.63 | 0.59 |
| n1-standard-4 (GCE, europe-west1-c) | 0.75 | 0.59 | 0.60 | 0.66 | 0.60 | 0.66 | 0.83 | 0.82 | 1.00 | 0.62 | 0.63 | 0.65 | 0.61 |
| n1-highcpu-2 (GCE, europe-west1-c) | 0.55 | 0.53 | 0.53 | 0.58 | 0.53 | 0.57 | 0.63 | 0.60 | 0.65 | 0.97 | 0.83 | 0.85 | 0.69 |
| n1-highmem-2 (GCE, europe-west1-c) | 0.54 | 0.52 | 0.53 | 0.58 | 0.52 | 0.58 | 0.62 | 0.61 | 0.65 | 0.82 | 0.98 | 0.85 | 0.68 |
| n1-standard-2 (GCE, europe-west1-c) | 0.52 | 0.49 | 0.50 | 0.70 | 0.49 | 0.55 | 0.59 | 0.58 | 0.62 | 0.81 | 0.83 | 0.98 | 0.68 |
| n1-standard-1 (GCE, europe-west1-c) | 0.52 | 0.53 | 0.48 | 0.55 | 0.53 | 0.53 | 0.62 | 0.55 | 0.62 | 0.70 | 0.65 | 0.68 | 1.00 |

## 5.4 *Evaluating the Distance Measure by Comparing Different Machines from Different Providers*

The sections above have shown that the proposed Euclidian distance measure is able to produce plausible results. Nevertheless, the distance measure was not developed to compare different machine types of the same provider (the measure is just cross checked against these uses cases to show that the results of the Euclidian distance are reasonable). The main purpose of this measure is to compare different machine types of different providers.

If different providers are compared, no sharp main diagonal can be excepted any longer. If that was the case, it would mean, that different cloud service providers would try to copy performance characteristics of competitors one by one. But of course, similarity clusters are expectable and should be identified by a similarity measure. And these clusters should be aligned to some degree to the processing power of the instances. Of course, AWS and GCE are both providing 16, 8, 4, 2 thread machines. And 16 thread machines should show a somewhat similar and better performance than 8, 4, and 2 thread machines.
Table 3 shows measured Euclidian similarity values for various machine pairs of AWS and GCE. A first '*16 thread cluster*' consists of the '`*.4xlarge`' AWS and the '`n1-*-16`' GCE machine types. A '*8 thread cluster*' consists of the '`*.2xlarge`' AWS and '`n1-*-8`' GCE machine types. The reader will identify easily 4 and 2 thread clusters as well and will even identify the small '*one thread cluster*' consisting of the AWS '`m3.medium`' and GCE '`n1-standard-1`' low-cost machine pair.
So the similarity values of the Euclidian distance measure are also reasonable for the intended use case to compare different machine types from different providers and the similarity values are aligned to cloud engineer expectations. Even more interesting, the Euclidian similarity is also telling something about machine types which are more similar than others – and therefore more preferable for container clusters. Table 3 shows three pairs of AWS and GCE machine types which show more similarity than other combinations (similarities of 0.97 and 0.98 which is comparable to inner GCE similarities between datacenter, see Table 2). These pairs would be a first choice for a container cluster.

**Table 3**: Similarity values of various AWS and GCE machines types (sorted on both axis by descending amount of simultaneous executable threads)

| Similarities | n1-highcpu-16 (GCE) | n1-highmem-16 (GCE) | n1-standard-16 (GCE) | n1-highcpu-8 (GCE) | n1-highmem-8 (GCE) | n1-standard-8 (GCE) | n1-highcpu-4 (GCE) | n1-highmem-4 (GCE) | n1-standard-4 (GCE) | n1-highcpu-2 (GCE) | n1-highmem-2 (GCE) | n1-standard-2 (GCE) | n1-standard-1 (GCE) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| c3.4xlarge (AWS) | 0.80 | 0.83 | 0.78 | 0.64 | 0.64 | 0.74 | 0.57 | 0.52 | 0.53 | 0.51 | 0.44 | 0.46 | 0.46 |
| i2.4xlarge (AWS) | 0.81 | 0.84 | 0.79 | 0.66 | 0.66 | 0.61 | 0.60 | 0.54 | 0.56 | 0.53 | 0.47 | 0.49 | 0.48 |
| r3.4xlarge (AWS) | 0.80 | 0.83 | 0.78 | 0.64 | 0.65 | 0.60 | 0.57 | 0.52 | 0.54 | 0.51 | 0.45 | 0.46 | 0.46 |
| c3.2xlarge (AWS) | 0.81 | 0.61 | 0.65 | 0.82 | 0.78 | 0.82 | 0.61 | 0.61 | 0.77 | 0.53 | 0.52 | 0.54 | 0.51 |
| i2.2xlarge (AWS) | 0.66 | 0.61 | 0.65 | 0.82 | 0.78 | 0.83 | 0.61 | 0.62 | 0.63 | 0.53 | 0.53 | 0.54 | 0.52 |
| m3.2xlarge (AWS) | 0.66 | 0.61 | 0.65 | 0.82 | 0.78 | 0.97 | 0.61 | 0.62 | 0.63 | 0.53 | 0.53 | 0.54 | 0.52 |
| r3.2xlarge (AWS) | 0.66 | 0.61 | 0.65 | 0.82 | 0.78 | 0.82 | 0.61 | 0.62 | 0.63 | 0.53 | 0.53 | 0.54 | 0.52 |
| c3.xlarge (AWS) | 0.60 | 0.54 | 0.61 | 0.76 | 0.58 | 0.65 | 0.79 | 0.83 | 0.83 | 0.58 | 0.61 | 0.75 | 0.57 |
| i2.xlarge (AWS) | 0.59 | 0.53 | 0.60 | 0.61 | 0.57 | 0.79 | 0.78 | 0.82 | 0.83 | 0.57 | 0.61 | 0.60 | 0.56 |
| m3.xlarge (AWS) | 0.75 | 0.56 | 0.60 | 0.64 | 0.60 | 0.65 | 0.82 | 0.82 | 0.98 | 0.61 | 0.61 | 0.62 | 0.59 |
| r3.xlarge (AWS) | 0.60 | 0.54 | 0.60 | 0.61 | 0.58 | 0.79 | 0.79 | 0.83 | 0.84 | 0.59 | 0.62 | 0.62 | 0.58 |
| c3.large (AWS) | 0.52 | 0.46 | 0.53 | 0.51 | 0.49 | 0.56 | 0.57 | 0.64 | 0.63 | 0.77 | 0.84 | 0.81 | 0.79 |
| m3.large (AWS) | 0.54 | 0.47 | 0.55 | 0.68 | 0.50 | 0.58 | 0.59 | 0.64 | 0.64 | 0.80 | 0.85 | 0.97 | 0.67 |
| r3.large (AWS) | 0.65 | 0.45 | 0.53 | 0.51 | 0.48 | 0.56 | 0.57 | 0.64 | 0.76 | 0.77 | 0.83 | 0.81 | 0.65 |
| m3.medium (AWS) | 0.38 | 0.34 | 0.41 | 0.39 | 0.36 | 0.42 | 0.43 | 0.48 | 0.47 | 0.51 | 0.57 | 0.55 | 0.85 |

The presented data was double checked by comparing same machine types of different datacenters of the same provider. We did this for AWS and GCE. Due to page limitations, we presented only data for GCE. Nevertheless, all collected data for AWS and GCE come to the same conclusion. Our proposed similarity measure is in accordance with cloud service engineer's expectation and in accordance with Lin's difference and identical intuition of a good similarity measure.

# 6. Conclusion

Cloud service selection can be a complex and challenging task for a cloud engineer. The underlying decision making problem makes it difficult to model it appropriately. Furthermore, very often cloud service selection is not done very systematically in practice. Using a 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 CoreOS, Kubernetes or Mesos. 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. As Section 3 showed, there exist several approaches to compare public cloud service providers. Most of these approaches focus to select best-performing and cost-effective providers but not to select similar resources offered by different providers. This contribution concentrated on finding most similar resources and not the most performant or cost-effective one's.

EasyCompare is able to determine performance differences of different virtual machine types as a numeric value between 0 (no similarity) and 1 (maximum similarity). This is done by calculating a normalized Euclidian distance measure. We evaluated the distance measure and showed that it is relevant, able to identify similar resources and provides reasonable similarity values according to Lin's intuitions of a good similarity measure. We analyzed other similarity measures like cosine similarity, dice similarity, euclidian distance and euclidian distance metric as well, but found no other appropriate measures to express virtual machine similarity in an appropriate way.

Comparing virtual machines of different cloud service providers is mainly a combinatoric problem. It turned out, that not hundreds of theoretical possible machine pairs have to be considered in practice, but only three to five with relevant similarities. These three to five relevant machine types can be identified by systematic benchmark runs in less than three hours. EasyCompare is designed to do fast and completely automatic benchmarking by only collecting data relevant to identify such similarities. We analyzed AWS and GCE and it turned out that only three machine pairs of these both providers (out of 195 analyzed machine pairs) are showing high similarities (see Section 5.4). All other machine types (that is more than 98.5% of the problem space) are of minor relevance for cloud engineers dealing with container clusters. Similar effects for other cloud service providers are more than likely. Identifying most appropriate service providers for such a limited set of resources is obviously much simpler than comparing complete cloud service provider resource offerings.

# Acknowledgement

This study was funded by German Federal Ministry of Education and Research (03FH021PX4, Cloud TRANSIT). We used research grants by courtesy of Amazon Web Services and Google Compute Engine. The authors thank Lübeck University (Institute of Telematics) and fat IT solution GmbH (Kiel) for their general support of the research project Cloud TRANSIT.

# References

[1] A. Afshari, M. Mojahed, and R. M. Yusuff. Simple additive weighting approach to personnel selection problem. *International Journal of Innovation, Management and Technology*, 1(5):511–515, 2010.

[2] Apache. Cassandra. http://cassandra.apache.org/. Accessed May 20, 2015.

[3] Apache. Hadoop. http://hadoop.apache.org/. Accessed May 20, 2015.

[4] Apache. Hdfs. http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html. Accessed May 20, 2015.

[5] Apache. Mahout. http://mahout.apache.org/. Accessed May 20, 2015.

[6] Apache. Mesos. http://mesos.apache.org. Accessed May 20, 2015.

[7] G. Ataş and V. C. Gungor. Performance evaluation of cloud computing platforms using statistical methods. *Comput. Electr. Eng.*, 40(5):1636–1649, July 2014.

[8] T. Bray. Bonnie benchmark, 1996. http://www.textuality.com/bonnie/. Accessed May 20, 2015.

[9] 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*, pages 199–208. IEEE, 2012.

[10] C. Chen, P. Maniatis, A. Perrig, A. Vasudevan, and V. Sekar. Towards verifiable resource accounting for outsourced computation. *SIGPLAN Not.*, 48(7):167–178, Mar. 2013.

[11] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[12] CoreOS. Coreos. https://coreos.com. Accessed May 20, 2015.

[13] Dato. Dato graphlab create. https://dato.com/products/create/open_source.html. Accessed May 20, 2015.

[14] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[15] E. M. B. C. (EEMBC). Eembc – the embedded microprocessor benchmark consortium. https://www.eembc.org/coremark/. Accessed May 20, 2015.

[16] 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.*, 64(3):835–848, June 2013.

[17] 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*, PP(99):1–1, 2015.

[18] faban.org. Faban - helping measure performance. http://faban.org/index.html, 2014. Accessed May 20, 2015.

[19] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. *SIGPLAN Not.*, 47(4):37–48, Mar. 2012.

[20] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '12, pages 37–48, New York, NY, USA, 2012. ACM.

[21] B. Fitzpatrick. Distributed caching with memcached. *Linux journal*, 2004(124):5, 2004.

[22] W. B. Frakes and R. Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

[23] 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*, pages 210–218. IEEE, 2011.

[24] Google. Kubernetes. http://kubernetes.io. Accessed May 20, 2015.

[25] Google. Perfkit benchmarker. https://github.com/GoogleCloudPlatform/PerfKitBenchmarker, 2015. Accessed May 20, 2015.

[26] J. L. Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[27] 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*, NSDI'11, pages 295–308, Berkeley, CA, USA, 2011. USENIX Association.

[28] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 41–51. IEEE, 2010.

[29] 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*, pages 341–348. IEEE, 2013.

[30] 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*, pages 410–413. IEEE, 2012.

[31] D. E. Knuth and D. Bibby. *The texbook*, volume 1993. Addison-Wesley Reading, MA, USA, 1986.

[32] N. Kratzke. Lightweight virtualization cluster - howto overcome cloud vendor lock-in. *Journal of Computer and Communication (JCC)*, 2(12), oct 2014.

[33] N. Kratzke. A lightweight virtualization cluster reference architecture derived from open source paas platforms. *Open Journal of Mobile Computing and Cloud Computing (MCCC)*, 1(2):17–30, 2014.

[34] A. Li, X. Yang, S. Kandula, and M. Zhang. Cloudcmp: Comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 1–14, New York, NY, USA, 2010. ACM.

[35] D. Lin. An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 296–304, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[36] P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi. The hpc challenge (hpcc) benchmark suite. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 213. Citeseer, 2006.

[37] Mark Gates, Alex Warshavsky. Iperf. https://iperf.fr/. Accessed May 20, 2015.

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

[39] H. McGhan. Spec cpu2006 benchmark suite. *Microprocessor Report*, 2006.

[40] M. McGill. An evaluation of factors affecting document ranking by information retrieval systems., 1979. Accessed May 20, 2015.

[41] M. Menzel and R. Ranjan. Cloudgenius: Decision support for web server cloud migration. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 979–988, New York, NY, USA, 2012. ACM.

[42] M. S. Müller, M. van Waveren, R. Lieberman, B. Whitney, H. Saito, K. Kumaran, J. Baron, W. C. Brantley, C. Parrott, T. Elken, et al. Spec mpi2007—an application benchmark suite for parallel systems using mpi. *Concurrency and Computation: Practice and Experience*, 22(2):191–205, 2010.

[43] S. Newman. *Building Microservices - Designing Fine-Grained Systems*. O'Reilly, 2015.

[44] W. D. Norcott and D. Capps. Iozone filesystem benchmark. *www.iozone.org*, 55, 2003.

[45] E. Parallel Systems Architecture Lab. Cloudsuite benchmarks. http://parsa.epfl.ch/cloudsuite/overview.html. Accessed May 20, 2015.

[46] Phoronix Media. Phoronix test suite. http://www.phoronix-test-suite.com/. Accessed May 20, 2015.

[47] R. Pozo and B. Miller. Java scimark 2.0. http://math.nist.gov/scimark2/. Accessed May 20, 2015.

[48] 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*, pages 21–26. ACM, 2013.

[49] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(1):17–30, 1989.

[50] K. Raman. Optimizing memory bandwidth on stream triad. https://software.intel.com/en-us/articles/optimizing-memory-bandwidth-on-stream-triad, 2013. Accessed May 20, 2015.

[51] Rick Jones . Network performance benchmark netperf. http://www.netperf.org/netperf/. Accessed May 20, 2015.

[52] H. Schmid and A. Huber. Measuring a small number of samples, and the 3v fallacy: Shedding light on confidence and error intervals. *Solid-State Circuits Magazine, IEEE*, 6(2):52–58, Spring 2014.

[53] K. Shiv, K. Chow, Y. Wang, and D. Petrochenko. Specjvm2008 performance characterization. In *Computer Performance Evaluation and Benchmarking*, pages 17–35. Springer, 2009.

[54] J. Siegel and J. Perdue. Cloud services measures for global use: the service measurement index (smi). In *SRII Global Conference (SRII), 2012 Annual*, pages 411–415. IEEE, 2012.

[55] Smith, Ben and Grehan, Rick and Yager, Tom and Niemi, DC. byte-unixbench - a unix benchmark suite. https://code.google.com/p/byte-unixbench. Accessed May 20, 2015.

[56] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, and D. Patterson. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0. In *Proc. of CCA*, volume 8, 2008.

[57] R. Statistical Package. R: A language and environment for statistical computing. *Vienna, Austria: R Foundation for Statistical Computing*, 2009.

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

[59] L. Sun, H. Dong, F. K. Hussain, O. K. Hussain, and E. Chang. Cloud service selection: State-of-the-art and future research directions. *Journal of Network and Computer Applications*, 45(0):134 – 150, 2014.

[60] A. Traeger, E. Zadok, N. Joukov, and C. P. Wright. A nine year study of file system and storage benchmarking. *Trans. Storage*, 4(2):5:1–5:56, May 2008.

[61] C. Vazquez, R. Krishnan, and E. John. Cloud computing benchmarking: A survey. In *Proceedings of International Conference on Grid and Cloud Computing and Applications (GCA 2014)*, 2014.

[62] S. Wu, Y. Bi, and X. Zeng. Using the euclidean distance for retrieval evaluation. In A. Fernandes, A. Gray, and K. Belhajjame, editors, *Advances in Databases*, volume 7051 of *Lecture Notes in Computer Science*, pages 83–96. Springer Berlin Heidelberg, 2011.

[63] J. Yang, W. Lin, and W. Dou. An adaptive service selection method for cross-cloud service composition. *Concurrency and Computation: Practice and Experience*, 25(18):2435–2454, 2013.

[64] L. Yi and J. Dai. Experience from hadoop benchmarking with hibench: from micro-benchmarks toward end-to-end pipelines. In *Advancing Big Data Benchmarks*, pages 43–48. Springer, 2014.

# Appendix

**Table 4**: Measured median performance values of several AWS and GCE machine types (sorted by descending processing performance)

| Machine Type | Threads | Memory (MB) | Tachyon (s) | Memory (MB/s) | Network (MB/s) | I/O Write (MB/s) | I/O Read (MB/s) |
|---|---|---|---|---|---|---|---|
| c3.4xlarge (AWS) | 16 | 30720 | 7.92 n=33 | 31512.40 n=83 | 254.00 n=120 | 123.16 n=87 | 7140.05 n=85 |
| i2.4xlarge (AWS) | 16 | 123904 | 8.45 n=33 | 27910.25 n=90 | 255.50 n=120 | 121.00 n=96 | 5629.10 n=93 |
| r3.4xlarge (AWS) | 16 | 123904 | 8.47 n=33 | 31502.55 n=76 | 256.00 n=120 | 122.80 n=85 | 7269.05 n=84 |
| c3.2xlarge (AWS) | 8 | 15360 | 15.72 n=36 | 29873.50 n=80 | 129.00 n=120 | 122.99 n=96 | 6914.41 n=84 |
| i2.2xlarge (AWS) | 8 | 62464 | 16.79 n=36 | 29278.75 n=80 | 128.50 n=120 | 122.17 n=89 | 6923.51 n=93 |
| m3.2xlarge (AWS) | 8 | 30720 | 16.83 n=36 | 29433.30 n=80 | 126.00 n=120 | 121.63 n=91 | 7157.27 n=84 |
| r3.2xlarge (AWS) | 8 | 62464 | 16.85 n=36 | 29511.40 n=82 | 129.50 n=120 | 122.98 n=94 | 6926.79 n=84 |
| c3.xlarge (AWS) | 4 | 8192 | 31.38 n=36 | 20137.95 n=80 | 96.55 n=120 | 122.94 n=93 | 7018.64 n=96 |
| r3.xlarge (AWS) | 4 | 30720 | 33.52 n=36 | 18674.15 n=78 | 96.70 n=120 | 122.00 n=87 | 6656.78 n=87 |
| m3.xlarge (AWS) | 4 | 15360 | 33.55 n=36 | 19613.50 n=80 | 128.50 n=120 | 112.72 n=105 | 6782.94 n=84 |
| i2.xlarge (AWS) | 4 | 30720 | 33.60 n=36 | 20289.05 n=80 | 96.60 n=120 | 122.76 n=87 | 7709.90 n=84 |
| c3.large (AWS) | 2 | 4096 | 62.46 n=36 | 11523.00 n=78 | 67.80 n=120 | 68.41 n=90 | 7449.54 n=87 |
| m3.large (AWS) | 2 | 8192 | 66.96 n=36 | 10977.95 n=80 | 85.30 n=120 | 82.21 n=84 | 6868.19 n=84 |
| r3.large (AWS) | 2 | 15360 | 66.98 n=36 | 11562.40 n=80 | 68.00 n=120 | 67.80 n=88 | 7618.09 n=88 |
| m3.medium (AWS) | 1 | 4096 | 168.05 n=36 | 6631.10 n=103 | 42.10 n=120 | 41.61 n=84 | 3779.64 n=100 |
| n1-highmem-16 (GCE) | 16 | 105472 | 8.05 n=39 | 26836.40 n=98 | 282.50 n=150 | 87.09 n=105 | 5650.34 n=175 |
| n1-highcpu-16 (GCE) | 16 | 15360 | 8.07 n=36 | 27008.70 n=80 | 117.50 n=120 | 86.85 n=84 | 5205.79 n=138 |
| n1-standard-16 (GCE) | 16 | 60416 | 8.07 n=42 | 26827.45 n=100 | 71.55 n=150 | 86.90 n=105 | 5560.74 n=162 |
| n1-highcpu-8 (GCE) | 8 | 8192 | 16.15 n=45 | 25811.80 n=99 | 148.50 n=150 | 85.21 n=105 | 4978.00 n=168 |
| n1-highmem-8 (GCE) | 8 | 53248 | 16.18 n=45 | 25664.05 n=120 | 302.50 n=150 | 85.33 n=105 | 5526.72 n=167 |
| n1-standard-8 (GCE) | 8 | 30720 | 16.22 n=45 | 25416.80 n=98 | 81.15 n=150 | 85.30 n=105 | 5283.85 n=151 |
| n1-highmem-4 (GCE) | 4 | 26624 | 32.34 n=45 | 18857.30 n=100 | 58.95 n=150 | 83.60 n=105 | 5354.39 n=146 |
| n1-standard-4 (GCE) | 4 | 15360 | 32.81 n=45 | 18164.70 n=100 | 93.25 n=150 | 83.59 n=105 | 4950.54 n=167 |
| n1-highcpu-4 (GCE) | 4 | 3584 | 32.88 n=45 | 18484.30 n=100 | 166.50 n=150 | 85.39 n=105 | 4787.73 n=169 |
| n1-highmem-2 (GCE) | 2 | 13312 | 64.82 n=45 | 10195.00 n=100 | 69.20 n=150 | 81.13 n=105 | 5054.46 n=174 |
| n1-highcpu-2 (GCE) | 2 | 2048 | 65.98 n=45 | 10577.40 n=98 | 190.50 n=150 | 84.52 n=105 | 5045.38 n=172 |
| n1-standard-2 (GCE) | 2 | 8192 | 67.57 n=45 | 9967.35 n=106 | 103.00 n=150 | 80.96 n=105 | 4693.51 n=180 |
| n1-standard-1 (GCE) | 1 | 4096 | 84.99 n=51 | 10740.60 n=99 | 120.50 n=150 | 61.65 n=109 | 4621.90 n=177 |