# The Limit of Horizontal Scaling in Public Clouds

QINGYE JIANG, The University of Sydney
YOUNG CHOON LEE, Macquarie University
ALBERT Y. ZOMAYA, The University of Sydney

Public cloud users are educated to practice horizontal scaling at the application level, with the assumption that more processing capacity can be achieved by adding nodes into the server fleet. In reality, however, applications—even those specifically designed to be horizontally scalable—often face unpredictable scalability issues when running at scale. In this article, we study the limit of horizontal scaling in public clouds by identifying sources of such limitations and quantitatively measuring their impact on processing capacity. To this end, we develop *ScaleBench* as a distributed and parallel cloud-scale testing framework and propose a capacity degradation index (CDI) to describe the level of capacity degradation observed in our benchmark studies. We have conducted extensive experiments in four real public clouds to identify possible bottlenecks in compute, block storage, networking, and object storage. Further, we carry out large-scale experiments with a real-life video transcoding application on worker fleets with up to 3200 vCPU cores. Our experimental results provide the quantitative evidence on the limit of horizontal scaling in public clouds. This helps cloud users make better design decisions on horizontally scalable applications.

CCS Concepts: • **General and reference** → **Empirical studies**; **Performance**; • **Computer systems organization** → **Cloud computing**;

Additional Key Words and Phrases: Cloud computing, horizontal scaling, benchmark

**ACM Reference format:**
Qingye Jiang, Young Choon Lee, and Albert Y. Zomaya. 2020. The Limit of Horizontal Scaling in Public Clouds. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 5, 1, Article 6 (February 2020), 22 pages.
https://doi.org/10.1145/3373356

## 1 INTRODUCTION

Since the induction of Amazon Web Services (AWS) in 2006, public clouds have been steadily gaining market share in the enterprise computing resource market [22]. Many public clouds advertise the seemingly "unlimited" resource pool they have, as opposed to the limited computing resource an ordinary user would have access to in an on-premise environment. Along with the move to public clouds is the adoption of horizontal scaling as the recommended technique to handle dynamic workloads, such as web applications and batch processing jobs. Public cloud service providers also offer features such as auto scaling, automatically managing the nodes in the worker fleet to meet workload requirements while avoiding over-provisioning. There is a large body of literature on

Authors' addresses: Q. Jiang and A. Y. Zomaya, J12 - Computer Science Building, 1 Cleveland Street, Darlington NSW 2006, Australia; emails: qjiang@ieee.org, albert.zomaya@sydney.edu.au; Y. C. Lee, Room 209, 4 Research Park Drive, Macquarie Park NSW 2109, Australia; email: young.lee@mq.edu.au.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Association for Computing Machinery.
2376-3639/2020/02-ART6 $15.00
https://doi.org/10.1145/3373356



RIGHTSLINK

the performance characteristics of computing resource on public clouds. Most of them adopt the "performance variation" approach [38]—considering resource contention due to multi-tenancy on the underlying physical host level (noisy-neighbor effect). With the implicit assumption that more processing capacity can always be achieved by adding nodes into the worker fleet, benchmark and statistics methods are used to determine the performance variation pattern of the computing resources, which is then used to design horizontally scalable systems, with the goal to achieve predictable processing capacity at scale [6].

In reality, very little is known on the limit of the horizontal scaling technique in public clouds. Although it is commonly agreed that bottlenecks exist in all systems—including public clouds—regardless of their design and scale. Apart from such implicit awareness that the horizontal scaling technique can fail, there exists no quantitative study on when and how it will fail. In particular, if one runs a horizontally scalable application on a particular public cloud, will its capacity continue to grow as long as more nodes can be acquired from the cloud service provider? Or, will its capacity will stop growing at some point (i.e., cloud-scale bottlenecks), regardless of any further growth in the number of nodes in the server fleet? A more practical question is, can such cloud-scale bottlenecks actually be reached by an ordinary cloud user to the point that the horizontal scaling technique stops working? If yes, can such cloud-scale bottlenecks be detected so that cloud users can make proper design decisions as early as possible?

In this article, we attempt to answer these two questions by identifying sources of cloud-scale bottlenecks and quantitatively measuring their impact on the capacity of horizontally scalable applications. To this end, we develop *ScaleBench* as a distributed and parallel benchmark framework. ScaleBench is capable of generating substantial and sustainable workload on public clouds, simulating the resource consumption pattern of various horizontally scalable applications. In particular, it detects cloud-scale bottlenecks in compute unit, block storage, networking, and object storage. In addition, we use a real-life video transcoding application to demonstrate that the horizontal scaling technique can fail to gain more capacity when such cloud-scale bottlenecks are reached. The specific contributions of this article include the following:

- We use the parallel execution of the same single-node application on multiple nodes to simulate a perfectly horizontally scalable application. A combination of serial and parallel evaluations are used to compare the node level capacity of a single-node system and a multi-node system. We propose the concept of capacity degradation index (CDI) to describe the degree of capacity degradation at scale.
- We perform extensive empirical studies on four public clouds.[1] We observe significant capacity degradation in three of them. This confirms that on multiple public clouds, cloud-scale capacity bottlenecks not only exist, but also can be easily detected by an ordinary cloud user. With as little as 20 worker nodes, we observe up to 24%, 52%, 14%, and 90% capacity degradation in overall system performance, block storage, networking, and object storage, respectively.
- We conduct large-scale experiments using a real-life video transcoding application, where the largest worker fleet utilizes 3200 vCPU cores. We demonstrate that when the above-mentioned cloud-scale bottleneck is reached, the capacity of the horizontally scalable application stops growing regardless of the growth in the number of nodes.

[1]It should be emphasized that the intention of this article is not to compare the performance of different public clouds, although we inevitably need to obtain and report performance benchmark data. To avoid such a misunderstanding, we deemphasize the cloud service providers by referring to them as Cloud A (AWS in the us-east-1 region), Cloud B (Aliyun in the China East 2 region), Cloud C (Open Telekom Cloud in the eu-de region), and Cloud D (Huawei Cloud in the China East 1 region).



RIGHTSLINK

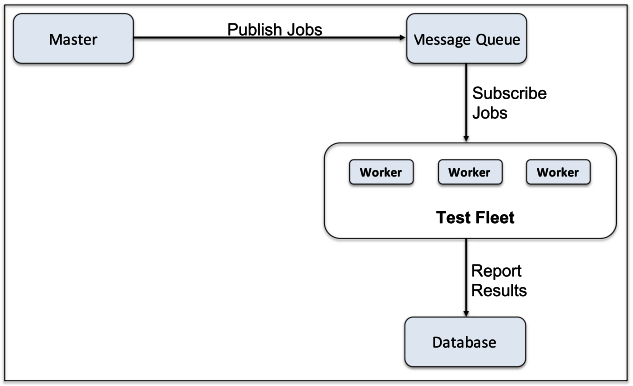


Fig. 1. The architecture of ScaleBench.

The rest of this article is organized as follows. Section 2 describes the design and implementation of ScaleBench. Section 3 describes the benchmarks on four basic building blocks of cloud-scale applications. In Section 4, we present and discuss the benchmark results obtained from four public clouds. In Section 5, we demonstrate how cloud-scale bottlenecks can impact the capacity growth of horizontally scalable applications, using a real-life large-scale video transcoding application as an example. Section 6 reviews related work, followed by our conclusions in Section 7.

## 2 SCALEBENCH

In this section, we present the architecture of ScaleBench and describe its implementation. We then propose CDI to quantitatively identify cloud-scale bottlenecks.

### 2.1 Architecture

In large-scale testing, we need to execute the same benchmark application with different runtime parameters on many nodes in parallel. As shown in Figure 1, ScaleBench employs a publisher/subscriber architecture, with four major components—the master node, the message queue, the worker nodes, and the database. The master node is the management node that publishes test jobs to the message queue. The worker nodes are the nodes being tested, and they subscribe to the message queue for test jobs to execute. Each worker node executes the test jobs locally, then generates a test report and writes the report to a database. The job submission application running on the master node does not need to wait for the test jobs to be completed. Further analysis can be carried out based on the test reports in the database.

There are two different types of jobs in ScaleBench—node-specific jobs and node-generic jobs. Node-specific jobs have node id's in the job definition. They are only executed on worker nodes with the node id's specified in the job. Node-generic jobs do not have node id's in the job definition. They are executed on all worker nodes. We use the private IP address of the worker node as its node id, which is unique in a private network. The test definition is described in JavaScript object notation (JSON) format. The attributes in the job description include (a) the node id, (b) the execution path (PATH) in which the command/script can be found, and (c) the actual command/script to execute. Multiple jobs can be packaged into a batch, within which they are executed in sequence according to the order they appear in the job description. Figure 2 is an example job definition with two jobs. In this example, the first command "df -h" is executed on all worker nodes, after that the worker node with IP address 172.31.0.20 executes the second command "ifconfig".



RIGHTSLINK

```
{
    "jobName": "demo",
    "jobs":[{
        "node": "*",
        "path": "/usr/sbin:/usr/bin:/sbin:/bin",
        "command": "df -h"
    },{
        "node": "172.31.0.20",
        "path": "/usr/sbin:/usr/bin:/sbin:/bin",
        "command": "ifconfig"
    }]
}
```

Fig. 2. An example of job definition in ScaleBench.

### 2.2 System Implementation

ScaleBench is implemented in Java.[2] The message queue is RabbitMQ, and the database is MySQL. In large-scale testing, a virtual machine image is created with the ScaleBench framework, and the related benchmark applications pre-installed and configured. The image is then used to launch identical worker nodes for testing. For automation, we take advantage of the cloud-init feature, which is commonly available on most public clouds. When the worker nodes are launched, the cloud-init script automatically starts the test agent, which subscribes to the message queue for jobs to execute. When new software packages or configuration/data files are needed on the worker nodes, the management node sends the instructions to perform the installation/download in the form of a sequence of test jobs.

Both RabbitMQ and MySQL run on the master node. The test agent on the worker nodes access RabbitMQ with a 1-second polling interval to fetch jobs to run. To ensure that there is only one job running at a time, the test agent stops polling RabbitMQ when a job is running. The test agent intercepts the job's stdout and stderr streams, and reports them back to MySQL as the test result. In our study, it takes minutes to hours to run benchmark jobs on the worker node. As such, RabbitMQ and MySQL are accessed only occasionally. The maximum workload is N requests per second, where N equals to the number of worker nodes.

### 2.3 Capacity Degradation Index (CDI)

The performance of a parallel system is usually measured by *speedup* (S) and *efficiency* (E). Speedup is defined as the ratio of the time needed to perform a certain amount of work on a single node ($T_1$) to the time needed to perform the same amount of work on $n$ nodes ($T_n$),

$$S(n) = \frac{T_1}{T_n} \tag{1}$$

Efficiency represents the average utilization of the $n$ nodes in a parallel system. Here, we make the assumption that a single-node system is fully utilized (efficient = 1). The relationship between efficiency and speedup is given by

$$E(n) = \frac{S_n}{n} \tag{2}$$

In benchmark studies, the *processing capacity* (C) of a system is usually not measured by how much time is needed to perform a certain amount of work, but rather how much work can be

[2] https://github.com/qyjohn/ScaleBench.



RIGHTSLINK

done in a certain amount of time. For example, the I/O capacity of a system is represented by its throughput in MB/s or GB/s. More complex system benchmark suites produce index scores, which are also derived from how much work can be done in a certain amount of time. For the same amount of work (W), we have

$$C(n) = \frac{W}{T_n} \tag{3}$$

Therefore,

$$S(n) = \frac{C_n}{C_1} \tag{4}$$

and

$$E(n) = \frac{C_n}{n \cdot C_1} \tag{5}$$

In ScaleBench, we use CDI to quantitatively describe how possible it is for a horizontally scalable application to encounter capacity degradation when running on a particular public cloud at scale. This is achieved by measuring the node level capacity difference between a single-node system (measured in the serial evaluation) and a multi-node system (measure in the parallel evaluations). In the serial evaluation, the same test is performed *n* times in sequence, while each test is performed on a different worker node (or a node pair when a particular test involves a pair of nodes). Each test generates the same workload on the worker node. In the parallel evaluation, the same test is performed on *n* nodes or node pairs in parallel. The worker nodes do not communicate with each other in any way, and the workload on the worker nodes does not depend on each other in any way. This simulates a perfectly horizontally scalable application with a combined workload, which is *n* times as big as the workload in the serial evaluation. We define CDI(n) as the node level capacity difference between the serial and parallel evaluations when there are *n* worker nodes or node pairs in the parallel evaluation. More formally,

$$CDI(n) = 100 \times \frac{C_s - C_p}{C_s} \tag{6}$$

$C_s$ is the average node level capacity measured in the serial evaluation,

$$C_s = \bar{C_1} \tag{7}$$

$C_p$ is the average node level capacity measured in the parallel evaluation,

$$C_p = \frac{C_n}{n} \tag{8}$$

Therefore,

$$CDI(n) = 100 \times (1 - E_n) \tag{9}$$

Throughout the rest of this article, we use a fixed *n* in our parallel evaluations and the notation CDI actually refers to CDI(n). Considering the seemingly unlimited size of the resource pool and the multi-tenant nature in public clouds, we can assume that the benchmark workload is only a very small portion of the total workload on the cloud. If noisy-neighbor effect on the physical host level (for example, virtual machines collocated on the same physical host) is the major source of performance variation, the node level capacity from the serial and parallel evaluations would be statistically similar, which is reflected in a small CDI. However, if in the parallel evaluation the combined workload reaches any of the cloud-scale bottlenecks, the node level capacity would become smaller due to resource contention, which is reflected in a relatively big CDI.

As such, CDI represents the average degree of capacity degradation on each worker node when there are *n* nodes (or node pairs) in the worker fleet. For example, if in a particular test $C_s$ is 100 and $C_p$ is 80, then $CDI = 20$ indicates each worker node loses 20% of its desired (or perceived) capacity



RIGHTSLINK

in the parallel evaluation. When *CDI* is small, it is unlikely to encounter capacity degradation when running horizontally scalable applications at scale. When *CDI* is big, it is likely to encounter capacity degradation when running horizontally scalable applications at scale. As compared to efficiency, CDI is more intuitive in describing capacity degradation in benchmark studies.

## 3 BENCHMARKS DESIGN

### 3.1 Overall System Performance

UnixBench 5.1.3[3] is used for benchmarking the overall system performance. UnixBench was started at Monash University in 1983 as a simple benchmark application. Later on, it was updated and revised by many people with new features and capabilities. UnixBench includes a number of individual tests such as Dhrystone, Whetstone, execl() throughput, file copy, pipe throughput, process creation, shell scripts, system call overhead, and graphical test. Each individual test was designed to evaluate the performance of different components on a computer system. The system benchmark index score reported by UnixBench reflects the overall performance of the computer system, which can be impacted by individual components including CPU, memory, and disk I/O.

UnixBench produces two system benchmark index scores—a single-thread system benchmark index score where the benchmark is performed in a single-thread fashion using a single CPU core, and a multi-thread system benchmark index score where the benchmark is performed in a multi-thread fashion using all available CPU cores on the system. In this article, we report the multi-thread system benchmark index score as the test result.

### 3.2 Block Storage

In public clouds, block storage is usually implemented as a separate service, provided to the compute service over the network. Certain public clouds have both volume-level throughput limits and instance-level throughput limits. In our test, we use 4 x 500 GB block devices to form a RAID0 disk array on each virtual machine instance to achieve the maximum throughput.

IOzone 3.471[4] is used for storage benchmark. IOzone evaluates file system capacity by generating a variety of file operations and reports the observed file I/O throughput (in MB/s). The test job generates a sustained level of sequential write operations by writing to a single test file on the RAID0 device in a single thread fashion. To account for the impact from memory and disk I/O buffer cache, the size of the test file is 200 GB, which is significantly larger than the amount of memory available on the worker node. The file system is Ext4 and the block size of each write operation is 256 KB. We format the file system with "mkfs.ext4 -E lazy_itable_init=0,lazy_journal_init=0 /dev/md0" to avoid the impact of Ext4 lazy initialization.

### 3.3 Networking

We develop our own network benchmark application to measure the sustainable throughput between a pair of worker nodes. On each worker node, we install the Apache web server with a 100 MB test file for downloading. Each worker node runs a Java program that repeatedly downloads the test file from the other worker node with multiple threads. In a single test, the test file is downloaded 3,200 times, resulting in 320 GB transmit/receive traffic on both nodes. Since the size of the test file is significantly smaller than the amount of memory available on the worker node, it is cached in memory when the first download completes so that there is no further disk I/O in subsequent downloads. Based on the time needed to finish the downloads, we calculate and report the average download throughput (in MB/s) observed on each node.

[3] https://github.com/kdlucas/byte-unixbench.
[4] http://www.iozone.org/.



RIGHTSLINK

```
{
    "jobName": "network",
    "jobs":[{
        "node": "172.31.0.21",
        "path": "/usr/sbin:/usr/bin:/Benchmark",
        "command": "bash nettest.sh 172.31.0.22"
    },{
        "node": "172.31.0.22",
        "path": "/usr/sbin:/usr/bin:/Benchmark",
        "command": "bash nettest.sh 172.31.0.21"
    }]
}
```

Fig. 3. The job definition to perform network benchmark between a pair of worker nodes 172.31.0.21 and 172.31.0.22.

Table 1. Worker Node Configurations

| | Instance Type | vCPU Cores | Memory (GB) | Root Vol. (GB) |
|---|---|---|---|---|
| Cloud A | c3.8xlarge | 32 | 60 | 500 |
| Cloud B | ecs.c5.8xlarge | 32 | 64 | 500 |
| Cloud C | s2.8xlarge.2 | 32 | 64 | 500 |
| Cloud D | s2.8xlarge.2 | 32 | 64 | 500 |

We use node-specific jobs in ScaleBench to automate the benchmark process. Figure 3 is an example job definition to perform networking test on two nodes 172.31.0.21 and 172.31.0.22.

### 3.4 Object Storage

In public clouds, object storage is usually implemented as a separate service. On each worker node, we run a Java program that repeatedly uploads a 100 MB test file to object storage, then deletes the file from object storage. In a single test, the test file is uploaded 3,200 times, resulting in 320 GB data transmission from the worker node to object storage, as well as 3,200 **PutObject** and **DeleteObject** API calls. Similar to the networking test, the test file is cached in memory when the first upload completes so that there is no further disk I/O in subsequent uploads. Based on the time needed to finish the test, we calculate and report the average upload throughput (in MB/s) observed on each node.

In public clouds, object storage is usually implemented with a distributed system in the back end, utilizing a hash algorithm to map object keys to storage partitions. In large-scale testing, improper naming for the object keys can cause the objects to be stored on only a few partitions in the distributed system. This is commonly referred to as the "hot partition" problem. The result is the over-utilization of the hot partitions, leading to performance degradation on the application side. In our test we generate a new universally unique identifier (UUID) as the object key for each upload, mitigating the above-mentioned "hot partition" problem.

## 4 RESULTS AND DISCUSSION

We performed our test on four selected public clouds—namely Cloud A, Cloud B, Cloud C, and Cloud D—from February to October 2018. All worker nodes are virtual machines, not bare metal servers. Table 1 shows the configurations for the worker nodes. The master node is a separate virtual machine instance with the same configuration. On all public clouds, the worker nodes are



RIGHTSLINK

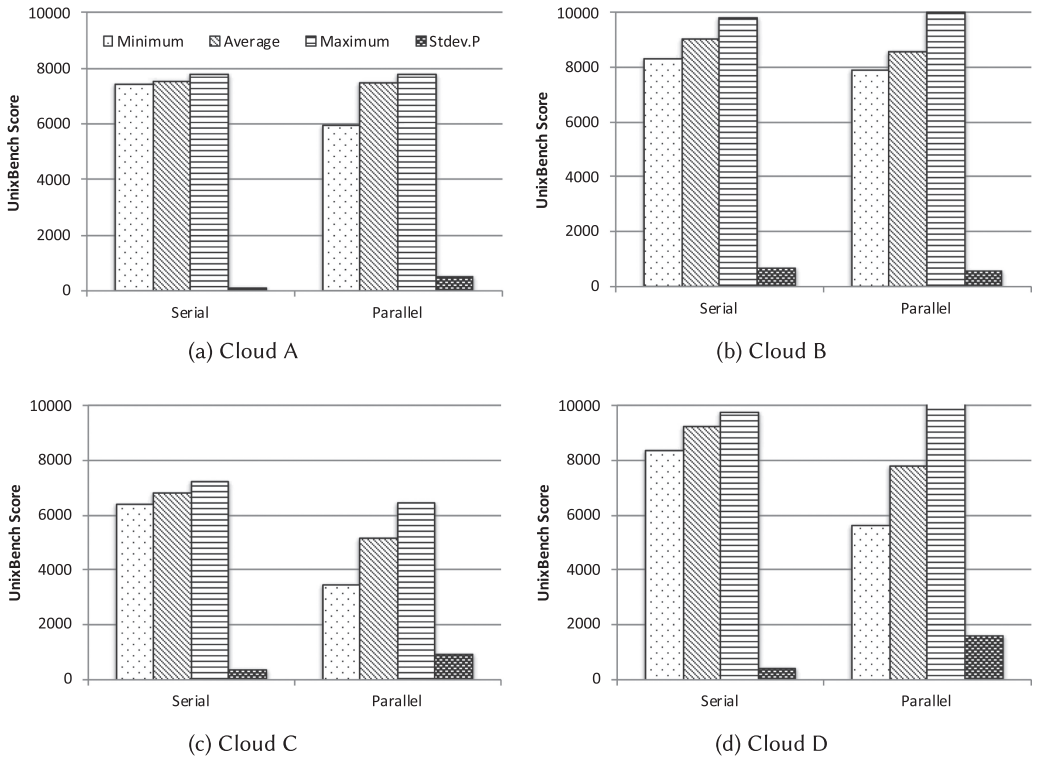


Fig. 4. Multi-thread UnixBench index score.

launched into the same availability zone.[5] The operating system is 64-bit Ubuntu 16.04.3 LTS. For each benchmark, we set n = 20 in both the serial and parallel evaluations.[6] In both evaluations, we report the minimum, average, and maximum capacity observed, along with the population standard deviation (Stdev.P).

With 20 worker nodes in the test fleet, both RabbitMQ and MySQL receive 20 requests per second at maximum. This is a very small workload, as compared to the processing capacity of the master node. On the worker nodes, we observe that (a) the average round-trip latency for requests to RabbitMQ and MySQL is within 10 milliseconds, and (b) the test agent consumes less than 0.5% CPU resources on a single vCPU core, with no disk I/O activity.

### 4.1 Overall System Performance

Figure 4 presents the multi-thread UnixBench system benchmark index scores. On Cloud A (Figure 4(a)), the average measurements from the serial and parallel evaluations are very similar, with the CDI being 0.7%. On Cloud B (Figure 4(b)), there is some modest capacity degradation in the

[5]In public clouds, there are usually multiple availability zones in a region. If we are able to detect cloud-scale bottlenecks on the availability zone level, it is reasonable to infer that the same can be detected on the region level by deploying worker nodes into other availability zones.

[6]On AWS, the default EC2 on-demand instance limit is 20 in all regions. Customers can request limit increases when needed. Other public clouds have similar default quotas and limit increase mechanisms. One of our research questions is whether cloud-scale bottlenecks can be reached by an ordinary cloud user. As such, it is desired for the cloud-scale bottlenecks to be detected with as little computing resource as possible. Therefore, we choose n = 20 in our benchmark tests. Evaluation at a larger scale is done in the case study part of this article.



RIGHTSLINK

parallel evaluation, with the CDI being 6%. On Cloud C (Figure 4(c)) and Cloud D (Figure 4(d)), there are obvious capacity degradation in the parallel evaluations, with the CDI's being 24% and 16%. For a horizontally scalable application, it is likely to encounter capacity degradation when running on Cloud C and Cloud D at scale. We also notice that in the parallel evaluations on Cloud C and Cloud D, the population standard deviation is high (>10%). This suggests that the worker nodes demonstrate a high level of capacity variation in the parallel evaluation—their overall system performance becomes less predictable.

For a virtual machine, CPU and memory resources are local resources on the underlying host server. In public clouds, it is a common practice to use the same type of CPU and memory for the same instance type. This is partially verified on all four public clouds by looking into the CPU information (/proc/cpuinfo) on the virtual machines. Virtualization technology allows cloud service providers to practice CPU over-commit (selling more virtual vCPU cores than the number of physical CPU cores available) or memory over-commit (selling more memory than the amount of physical memory available). In the serial evaluation, each test only utilizes 32 vCPU cores, which is relatively small as compared to the advertised "unlimited" resource pool in public clouds. The chance of the worker node being co-located with other instances—thus subjected to the impact of CPU/memory over-commit—is relatively small. In the parallel evaluation, we utilize 640 vCPU cores in parallel. For an established public cloud like AWS with a big resource pool, the chance of worker node co-location is expected to be small. For an emerging public cloud with a relatively small resource pool, the possibility of two or more worker nodes being co-located on the same host server might be higher. When co-location occurs, the co-located worker nodes might suffer from the impact of CPU/memory over-commit, leading to the observed capacity degradation. The same applies to any mentioning of worker node co-location in the rest of this article.

However, CPU/memory over-commit is not the only potential reason for the capacity degradation observed on Cloud C and Cloud D. The index score reported by UnixBench is also influenced by disk I/O. In the parallel evaluations, UnixBench is launched on all worker nodes at the same time. If the cloud service provider does not practice CPU/memory over-commit, then disk I/O is expected to start at approximately the same time on all worker nodes, resulting in a burst in the combined pressure on block storage. As such, block storage can also be a potential reason for the capacity degradation observed.

### 4.2 Block Storage

Figure 5 presents the write throughput on a RAID0 device using 4 x 500 GB block storage volumes. For Cloud A (Figure 5(a)), the average measurements from the serial and parallel evaluations are the same, with the CDI being 0%. For Cloud B (Figure 5(b)), there is modest capacity degradation in the parallel evaluation, with the CDI being 5%. For Cloud C (Figure 5(c)) and Cloud D (Figure 5(d)), there exists significant capacity degradation in the parallel evaluations, with the CDI's being 47% and 52%. For a horizontally scalable disk I/O intensive application, it is likely to encounter modest capacity degradation when running at scale on Cloud B, and significant capacity degradation when running at scale on Cloud C and Cloud D. Similar to the observations made in the overall system performance benchmark, in this benchmark, the population standard deviation is quite high (>10%) in the parallel evaluations on Cloud C and Cloud D. This suggests that the worker nodes demonstrate a high level of capacity variation in the parallel evaluation—their disk I/O capacity becomes less predictable.

There are three potential bottlenecks in block storage. The first potential bottleneck is the storage bandwidth of the physical host. When two or more virtual machines are co-located on the same physical host, improper Quality of Service (QoS) configurations might cause the virtual machines to compete for storage bandwidth. The second potential bottleneck is the aggregated bandwidth



RIGHTSLINK

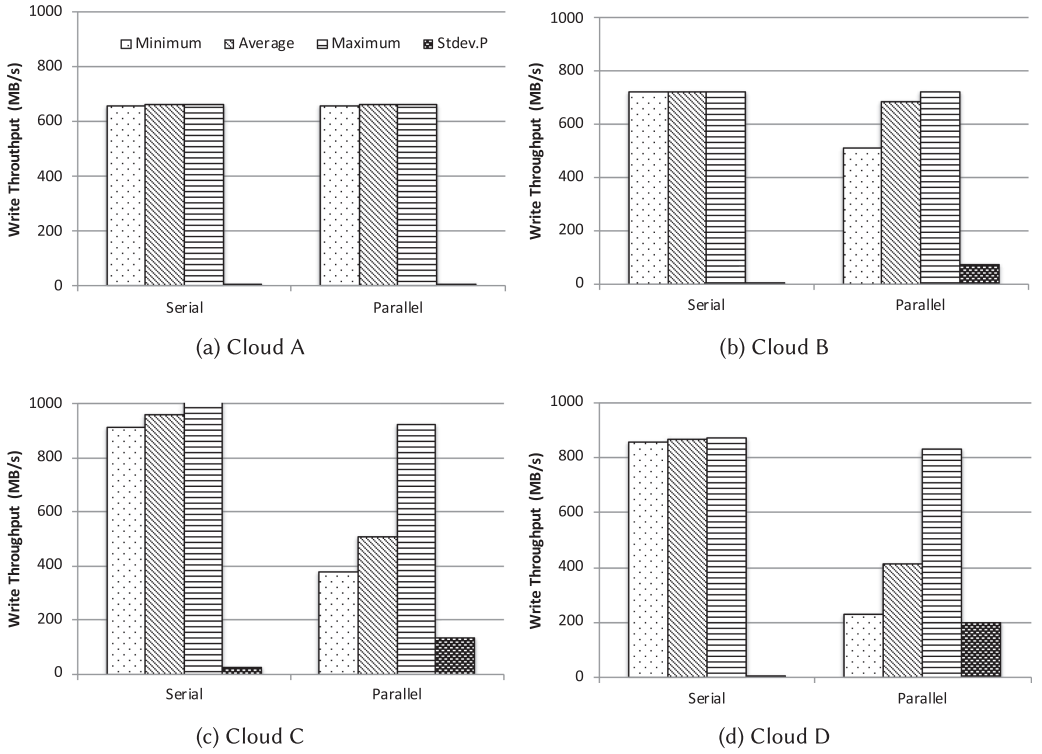


Fig. 5. 4 x 500 GB RAID0 disk write throughput (MB/s).

between compute service and block storage. When the combined workload reaches the aggregated bandwidth, higher throughput can not be achieved. In the parallel evaluation, the throughput on some worker nodes is approximately 1/2 or 1/4 of the throughput in the serial evaluation. This suggests that a single storage channel is then shared by 2 or 4 worker nodes. When a particular storage channel is saturated, capacity degradation occurs on worker nodes sharing that storage channel. The third potential bottleneck is the aggregated disk I/O capacity in the back end storage pool. The combined throughput observed in the parallel evaluation is 10,200 MB/s on Cloud C and 8,200 MB/s on Cloud D. For a public cloud service provider, the block storage service is expected to have much higher disk I/O capacity than these values. As such, in our benchmark, it is quite unlikely that the aggregated disk I/O capacity becomes the bottleneck.

Since cloud-scale bottleneck is observed in block storage on Cloud C and Cloud D, the previously observed capacity degradation (Figure 4(c) and (d)) in overall system performance is more likely to be related to block storage instead of CPU/memory over-commit.

### 4.3 Networking

Figure 6 presents the private network throughput between worker node pairs. For Cloud A (Figure 6(a)), Cloud B (Figure 6(b)), and Cloud C (Figure 6(c)), the average measurements in the serial and parallel evaluations are similar, with the CDI's being −1.5%, −0.2%, and 1%, respectively. For Cloud D (Figure 6(d)), there exists significant capacity degradation in the parallel evaluations, with the CDI being 14%. For a horizontally scalable network I/O intensive application, it is likely to encounter significant capacity degradation when running at scale on Cloud D.

Contrary to the observations made in overall system performance and block storage, in this benchmark, the population standard deviation is approximately the same in the serial and parallel



RIGHTSLINK

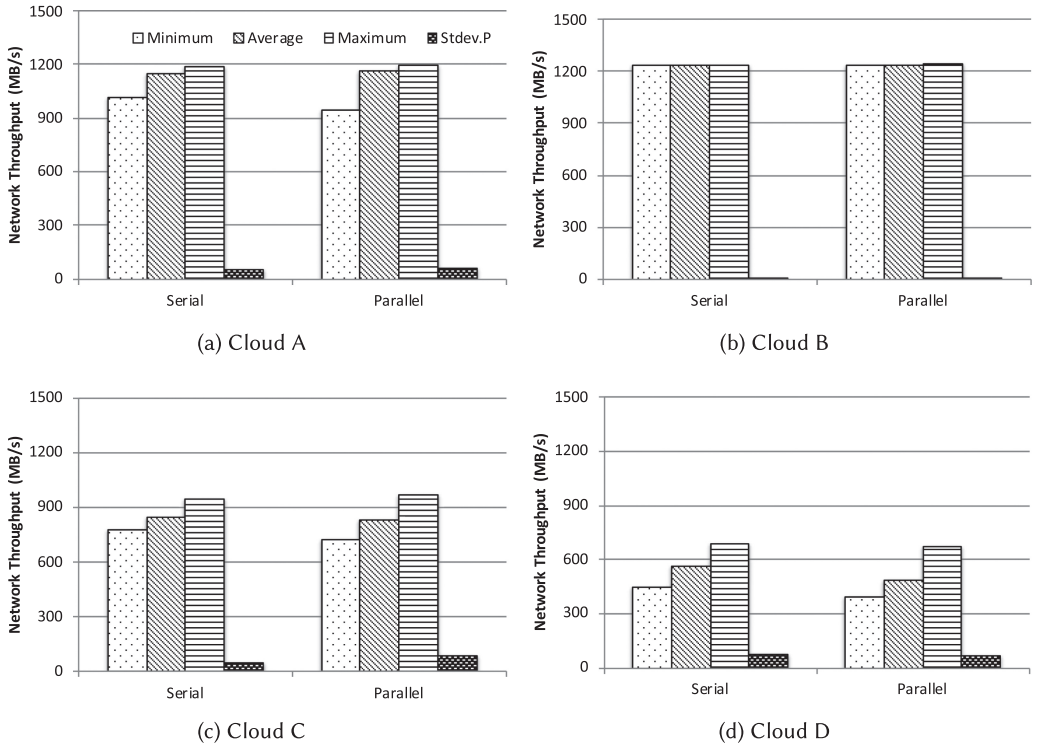


Fig. 6. Private network throughput between worker node pairs (MB/s).

evaluations on Cloud D, although significant capacity degradation is also observed. This suggests that on Cloud D, when the cloud-scale bottleneck on networking is reached, all worker nodes suffer from the same level of capacity lost. This also suggests that on Cloud D, the application network and the storage network might have different architectures, leading to the difference in the capacity degradation behavior.

There are two potential bottlenecks in networking. The first potential bottleneck is the network bandwidth of the physical host. When two ore more virtual machines are co-located on the same physical host, improper QoS configurations might cause the virtual machines to compete for network bandwidth, resulting in noisy-neighbor effect. The second potential bottleneck is the up-link bandwidth of the aggregate switches in a tree topology, where there is a core-pod-rack hierarchy in the data center. The network cards on the physical hosts are connected to the top-of-rack switches, with up-links to the aggregate switches in the same pod, then with up-links to the core switch in the data center. For a pair of virtual machines on two different racks, the traffic traverses through the aggregate switches. For a pair of virtual machines in two different pods, the traffic traverses through the aggregate switches and the core switch. Most switches are designed for over-subscription, where the combined downstream bandwidth is greater than the combined up-link bandwidth. Capacity degradation arises when the cross-rack/pod traffic demands more throughput than the up-link bandwidth.

### 4.4 Object Storage

Figure 7 presents the object storage upload throughput. For Cloud A (Figure 7(a)), the results obtained from the serial and parallel evaluations are similar, with the CDI being 2%. For Cloud B



RIGHTSLINK

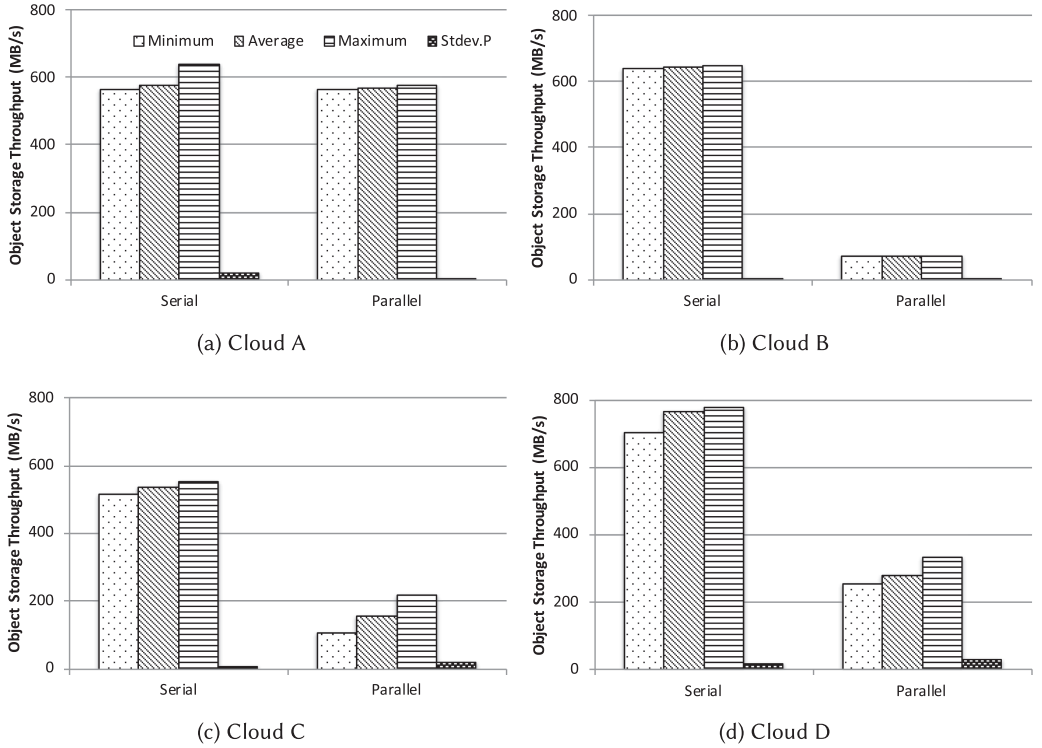


Fig. 7. Object storage upload throughput (MB/s).

(Figure 7(b)), Cloud C (Figure 7(c)), and Cloud D (Figure 7(d)), there exists significant capacity degradation in the parallel evaluations, with the CDI's being 90%, 71%, and 63%, respectively. For a horizontally scalable object storage I/O intensive application, it is likely to encounter significant capacity degradation when running on Cloud B, Cloud C, and Cloud D at scale.

Similar to the observations made in the networking benchmark, in this benchmark, the population standard deviation is approximately the same in the serial and parallel evaluations on Cloud B, Cloud C, and Cloud D, where significant capacity degradation is observed. Since the object storage benchmark generates a significant level of sustained pressure on networking, the same factor might be contributing to the observed capacity degradation in networking and object storage.

There are several potential bottlenecks in the benchmark for object storage. The first potential bottleneck is networking within the virtual private cloud (VPC), which has been discussed in the networking benchmark. For Cloud A, Cloud B, and Cloud C, no obvious capacity degradation is observed in the networking benchmark. For Cloud D, in the serial evaluations, the object storage upload throughput observed (765 MB/s) is bigger than the throughput observed in the networking benchmark (565 MB/s). In the networking test, the workload is bi-directional—a pair of nodes download 320 GB data from each other, producing 320 GB in transmit and 320 GB in receive on both nodes. In the object storage benchmark, the workload is uni-directional with 320 GB in transmit on the worker node. We perform a set of uni-directional networking test with one node downloading 320 GB data from the other node, producing 320 GB in transmit on one node and 320 GB in receive on the other node. In the uni-directional networking test, the observed throughput is 800 MB/s, which is greater than the throughput observed in the bi-directional networking benchmark and the object storage benchmark. In the parallel evaluations, the average object storage upload



RIGHTSLINK

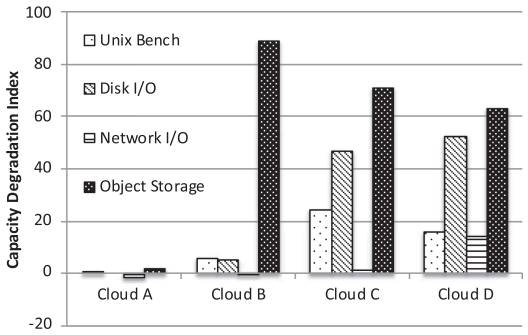


Fig. 8. Capacity degradation index (CDI).

throughput observed (282 MB/s) is smaller than the average throughput observed in the networking benchmark (486 MB/s). Therefore, networking within the VPC does not seem to be the bottleneck here.

The second potential bottleneck is the aggregated bandwidth between compute service and object storage, which might be an unpublished user quota. For Cloud B, the combined throughput in the parallel evaluation is approximately 1250 MB/s (10 Gbps). It is unlikely that public clouds still use switches with a 10 Gbps up-link. As such, we tend to believe this is an unpublished user quota. For Cloud C, the combined throughput in the parallel evaluation is approximately 3125 MB/s (25 Gbps), which is a well-known throughput used in modern switches. It is possible that there is a 25 Gbps data link between compute service and object storage. For Cloud D, the combined throughput in the parallel evaluation is approximately 5640 MB/s (45 Gbps), which is in between the well-known switch throughput 40 Gbps and 50 Gbps. It is unlikely that a networking limit is reached in the parallel evaluation.

The third potential bottleneck is the API rate limit of the object storage service. Since we use a test file with a fixed size, the combined throughput might be the API rate limit times the fixed size. We perform several additional tests on Cloud B, Cloud C, and Cloud D with smaller test files. On all three of them, we are able to make more API calls and observe approximately the same combined throughput. This indicates that API rate limit is not the bottleneck.

The four potential bottleneck is the processing capacity on the object storage service side. For Cloud D, the combined throughput observed is 5640 MB/s. This is not a significant value as compared to the disk I/O capacity of modern storage systems. As such, we tend to think that the underlying storage component is capable of providing sufficient disk I/O capacity, but the object storage service as a system is not capable of taking full advantage of the disk I/O capacity available.

### 4.5 Capacity Degradation Index

Figure 8 presents the CDI's obtained from all benchmarks, along with the raw data in Table 2. In some benchmarks, the CDI's are negative, suggesting capacity gain in the parallel evaluations. Such capacity gain is very small and can be considered as within the range of measurement errors. On Cloud A, no obvious capacity degradation is observed in any benchmark. On Cloud B, significant capacity degradation is observed in object storage. On Cloud C, modest capacity degradation is observed in overall system performance, significant capacity degradation is observed in block storage and object storage. On Cloud D, modest capacity degradation is observed in overall system performance and networking; significant capacity degradation is observed in block storage and object storage. This confirms that various cloud-scale bottlenecks exist in public clouds. Such bottlenecks can be easily detected by ScaleBench with as little as 20 worker nodes.



RIGHTSLINK

Table 2. Summary of Benchmark Results

| | Cloud A | Cloud B | Cloud C | Cloud D |
|---|---|---|---|---|
| Overall System Performance (UnixBench Index Score) | | | | |
| $M_s$ | 7,543 | 9,052 | 6,819 | 9,237 |
| $M_p$ | 7,490 | 8,554 | 5,157 | 7,775 |
| CDI(20) | 0.7 | 6 | 24 | 16 |
| 20 x $M_p$ | 149,800 | 171,080 | 103,140 | 155,500 |
| Block Storage Throughput (MB/s) | | | | |
| $M_s$ | 660 | 721 | 958 | 866 |
| $M_p$ | 660 | 684 | 508 | 413 |
| CDI(20) | 0 | 5 | 47 | 52 |
| 20 x $M_p$ | 13,200 | 13,680 | 10160 | 8,260 |
| Private Network Throughput (MB/s) | | | | |
| $M_s$ | 1,147 | 1,234 | 847 | 564 |
| $M_p$ | 1,165 | 1,237 | 835 | 486 |
| CDI(20) | −1.5 | −0.2 | 1 | 14 |
| 20 x $M_p$ | 23,300 | 24,740 | 16,700 | 9,720 |
| Object Storage Throughput (MB/s) | | | | |
| $M_s$ | 578 | 644 | 536 | 766 |
| $M_p$ | 568 | 62 | 156 | 282 |
| CDI(20) | 2 | 90 | 71 | 63 |
| 20 x $M_p$ | 11,360 | 1,240 | 3,120 | 5,640 |

$M_s$ represents the measurement from the serial evaluation. $M_p$ represents the measurement from the parallel evaluation.

The observed cloud-scale bottlenecks might be the result of architecture design, capacity limits in the resource pool, or unpublished user-level quota limits. In our studies, the selected public clouds are treated as black boxes. When attempting to understand these bottlenecks, we identify critical points in the data path and perform qualitative analysis around them. However, the discussion on the root cause of the observed bottlenecks is beyond the scope of this article. It should also be noted that such observations can change when the public cloud service providers upgrade their infrastructure or modify their QoS configurations. (On Cloud D, the results obtained in October 2018 are significantly better than the results obtained in February 2018. With the understanding that there might have been a significant upgrade in the service, we selectively report the better/later results in this article.)

When a very small CDI is observed in a particular benchmark, it does not mean that cloud-scale bottleneck does not exist. But rather, the proper interpretation is that cloud-scale bottleneck is not observed when there are only $n$ worker nodes in the parallel evaluation. For example, if in a particular compute resource pool (a rack or a cluster) there exists a shared channel to block storage with a 20 Gbps bandwidth and each virtual machine is subjected to a 1 Gbps throughput limit, then capacity degradation will not occur with 20 worker nodes but will start to occur when there are 21 or more worker nodes in the parallel evaluation.

It should be noted that we use some relatively big instance types (8xlarge) in our benchmark study, allowing us to observe significant performance degradation with as little as 20 worker nodes. As pointed out by Cortez et al. [7], small-size instance types are more commonly used in public clouds. With AWS, smaller instance types usually have lower network performance and storage bandwidth. Although not exactly the same, other public clouds have similar quality of service (QoS) measurements, allowing bigger instance types to consume more computing resources when



RIGHTSLINK

needed. As such, if the end user chooses to use small-size instance types in the benchmark, it might require a lot more worker nodes to arrive at similar observations as reported in this article.

## 5 CASE STUDY

The CDI data presented in Section 4.5 demonstrates the existence of cloud-scale bottlenecks in public clouds. In theory, it is possible for the horizontal scaling technique to fail when an application demands more resource than any of the cloud-scale bottlenecks. However, our second research question remains—can such cloud-scale bottlenecks actually be reached by an ordinary cloud user to the point that the horizontal scaling technique stops working? In this section, we attempt to answer this question with a case study.

As shown in Figure 8 and Table 2, the CDI's on overall system performance and networking are relatively lower than the CDI's on disk I/O and object storage. To demonstrate the impact of cloud-scale bottlenecks with as little computing resources as possible, it is desired that the demo application has an intensive demand on either disk I/O or object storage.

There exist three major types of horizontally scalable workloads on public clouds, namely multi-tier web service, batch processing, and big data analysis using Apache Hadoop or Apache Spark. In general, multi-tier web service workload presents more pressure on overall system performance or networking instead of disk I/O or object storage, making it an unfavorable use case for this study. The resource consumption of a batch processing system depends on the workload inside a batch. If the workload inside a batch is either disk I/O intensive or object storage intensive, then it can become a good use case for this study. The same applies to big data analysis workload—if the analysis is either disk I/O intensive or object storage intensive, then it can become a good use case for this study.

In this section, we choose to use a batch processing system for video transcoding as the use case. Video transcoding workload is well-known for being CPU intensive and disk I/O intensive at the same time. When the input source and output destination are on object storage, it can present an intensive pressure on object storage as well. With the understanding that video transcoding might not be a representative workload in public clouds, it has the potential of reaching some of the cloud-scale bottlenecks with as little computing resource as possible.

### 5.1 Video Transcoding Application

Video transcoding is a common use case for websites providing video sharing service. When an end user uploads a video for sharing, the uploaded video comes with a particular file format, resolution, and bitrate. To provide the best user experience for the viewers, the service provider often needs to convert the same video into different file format, resolution, and bitrate so that it can be delivered to different devices with different runtime environments over different network connectivity conditions.

Figure 9 presents the architecture of the video transcoding system used by a video sharing website [23]. When the end user uploads a video for sharing, the uploaded video is stored in a bucket on object storage. The upload API call also triggers an event notification action, which publishes a job to the message queue. The video transcoding fleet includes a group of worker nodes, which poll the corresponding message queue for transcoding jobs to execute. As such, the video transcoding system is a horizontally scalable batch processing system with a producer/consumer architecture. In this case study, we use a 5.26 MB video in MP4 format as the user upload, simulating a short video with a smart phone for sharing. We pre-stage 100,000 copies of the same video to object storage, with a UUID as the object key. On each worker node, the video transcoding application launches multiple threads, with the number of threads equal to the number of vCPU cores on the worker node. Each thread performs the following tasks: (a) receives one job from the message



RIGHTSLINK

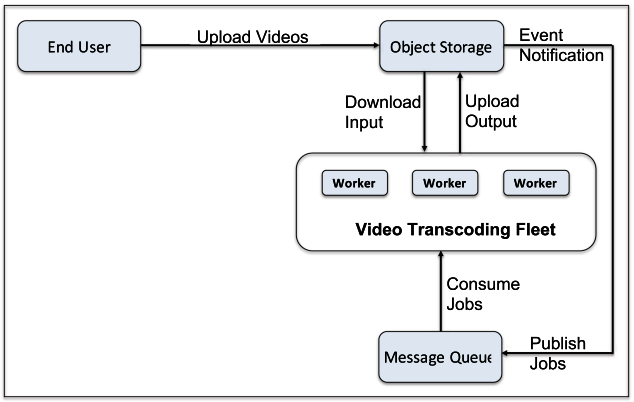


Fig. 9. The architecture of the video transcoding system.

queue, (b) downloads the corresponding video from object storage to local disk, (c) uses FFmpeg[7] to convert the video from MP4 format to WMV format (the output size is 5.76 MB), (d) uploads the output file from local disk to object storage, and (e) deletes both the MP4 and WMV files from local disk. This process continues until all jobs in the message queue have been consumed. In a production environment, the video transcoding fleet is deployed in an auto-scaling group, which changes its size based on the number of remaining jobs (queue length) in the message queue. In our evaluation, we manually specify the desired size of the worker fleet, instructing auto scaling to launch or terminate worker nodes to arrive at the desired number of worker nodes in the worker fleet for each test.

Video transcoding is a compute-intensive workload. Working with local files produces pressure on disk I/O. Using object storage as the initial input and final output produces pressure on both object storage and networking. In our experiments, we change the number of worker nodes and record the rate of transcoding (the number of videos transcoded per second) of the worker fleet as the processing capacity. As the number of worker nodes increases, the rate of video processing is expected to increase accordingly. However, as the scale of the worker fleet grows, certain cloud-scale bottlenecks might be reached. If this happens, the rate of transcoding would stop growing at some point, regardless of the increase in the number of worker nodes.

### 5.2 Resource Consumption Pattern

Figure 10 shows the resource consumption pattern of the video transcoding application running on a single worker node on Cloud A over a 50-second period. On Cloud B, Cloud C, and Cloud D, the application demonstrates a similar resource consumption pattern. As shown in Figure 10(a), the application demands almost 100% CPU most of the time (user), with only occasional resource under-utilization (idle). Disk I/O (Figure 10(b)) and network I/O (Figure 10(c)) occur in a burst pattern. (We omit disk reads from Figure 10(b) because it is very low.) The peaks in disk I/O relate well with the appearance in CPU under-utilization. This is because the 32 worker threads are working on different copies of the same video. They download the videos at the same time, spend the same amount of time on transcoding, and upload the output to object storage at the same time. The peaks in disk and network I/O are well below the corresponding single-node capacities observed in Sections 4.2 and 4.3. On average, the application demands 15–20 MB/s throughput in disk writes, network receive, and transmit. Out of the 60 GB memory available on the worker node,

[7] https://www.ffmpeg.org.



RIGHTSLINK

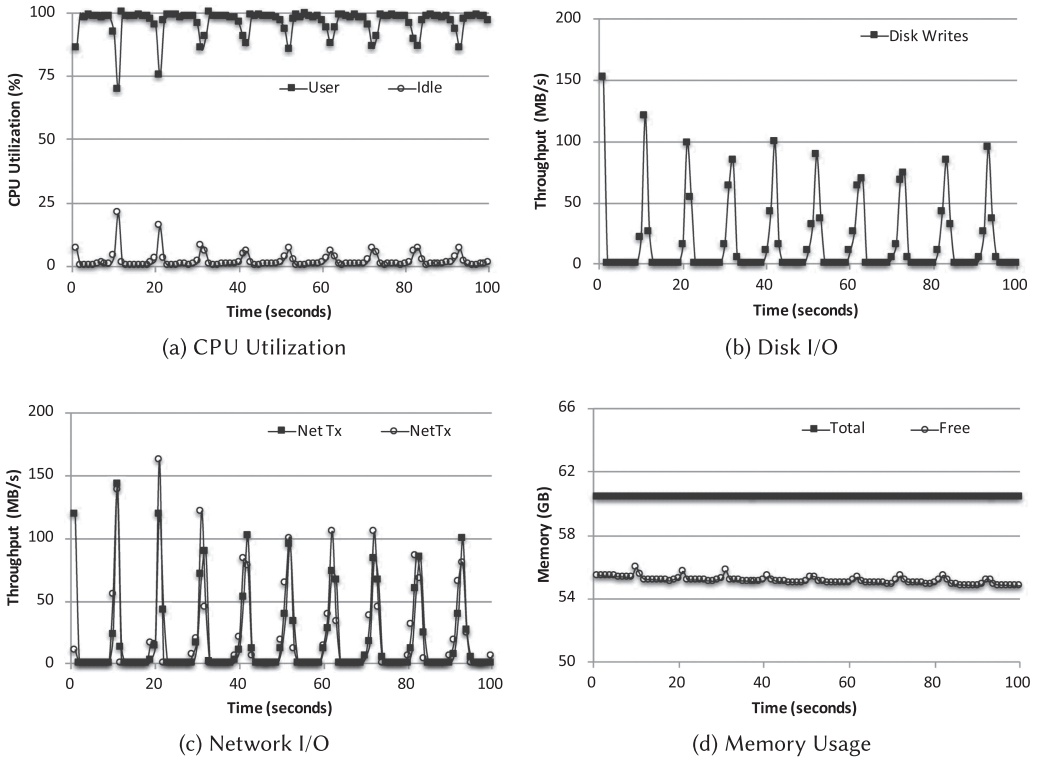


Fig. 10. Resource consumption pattern of the video transcoding application running on a single worker node.

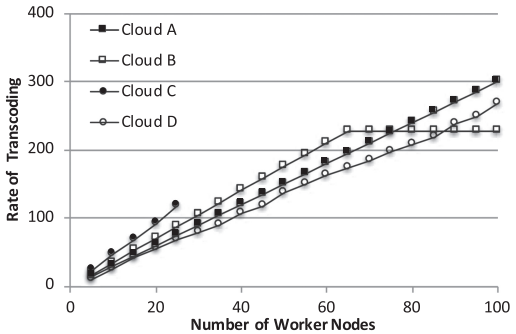


Fig. 11. The rate of transcoding vs. worker nodes.

around 55 GB is free. The video transcoding application can be considered as CPU-intensive, with relatively low demand on memory, disk, and network I/O. This makes it an ideal use case for the horizontal scaling technique.

### 5.3 Large-Scale Evaluation Results

Figure 11 presents the rate of video transcoding obtained from our large-scale experiments. On Cloud A, Cloud B, and Cloud D, the worker fleet has up to 100 worker nodes, with a maximum number of 3200 vCPU cores. On Cloud C, the test fleet has up to 25 worker nodes due to quota limits from the cloud service provider.



RIGHTSLINK

On Cloud A, there is a linear relationship between the rate of transcoding and the number of worker nodes, with no obvious signs of capacity degradation. This is expected because no cloud-scale bottleneck is detected on Cloud A in any benchmarks. On Cloud B, the linear relationship is observed with up to 65 worker nodes, beyond which no further capacity growth is observed. The maximum rate of transcoding is 227 videos per second, which demands 1203 MB/s in network throughput. This is very close to the object storage throughput limit (1250 MB/s) detected in Section 3.4. In this test, when the combined workload on object storage exceeds the cloud-scale bottleneck, further capacity can not be achieved by adding nodes into the worker fleet.

On Cloud C, such linear relationship is observed and maintained throughout our tests. Despite the fact that cloud-scale bottlenecks are identified in block storage, networking, and object storage in the benchmarks, none of these bottlenecks are reached in this case study because the small size of the worker fleet. With the previously-detected 3125 MB/s throughput limit in object storage, capacity degradation might not occur until the rate of transcoding reaches 590 videos per second. On a single node, the rate of transcoding is approximately 4.8 videos per second. It would require approximately 125 nodes in the worker fleet to reach the throughput limit in object storage.

On Cloud D, such linear relationship is observed and maintained throughout our tests. Despite the fact that cloud-scale bottlenecks have been identified in block storage, networking, and object storage in the benchmarks, none of these bottlenecks are reached in this evaluation. With the previously-detected 5640 MB/s throughput limit in object storage, capacity degradation might not occur until the rate of video transcoding reaches 1,065 videos per second. On a single node, the rate of transcoding is approximately 2.5 videos per second. It would require approximately 430 nodes in the worker fleet to reach the throughput limit in object storage.

### 5.4 General Use Cases

With the understanding that cloud-scale bottlenecks might prevent horizontally scalable applications from achieving the desired level of capacity in public clouds, we recommend the following procedure to make the proper design decisions:

- Use the serial evaluation technique to observe the resource consumption pattern of the application running on a single node. This includes a variety of capacity metrics such as disk I/O (input/output operations per second (IOPS) and throughput), network I/O (packets per second and throughput), and object storage (number of API calls per second and throughput). This becomes the desired single-node capacity $C_1$ for each capacity metric on a single node. The standard deviation observed in the serial evaluation is the expected level of capacity variation of the nodes. On average, a single node is capable of processing $n$ requests (jobs, tasks) per second. If your goal is to process $m$ requests per second, then you will need a total number of $t = m / n$ nodes in the worker fleet.
- Use the parallel evaluation technique to measure the various node level capacity metrics of a test fleet with $n$ worker nodes. Calculate the CDI's for each capacity metric. The larger the CDI is, the more possible it is for the horizontal scaling technique to fail. Also, calculate the combined capacity for each capacity metrics, denoted by $C_n$.
- For a particular performance metric, if $C_n < t * C_1$, then there exists a cloud-scale bottleneck that prevents the worker fleet from achieving the desired single-node capacity $C_1$ for all of the $t$ worker nodes. In this case, more processing capacity can not be achieved by adding nodes into the worker fleet and the horizontal scaling technique fails.

Public cloud service providers can also take advantage of ScaleBench to quickly identify cloud-scale bottlenecks in their services.



RIGHTSLINK

## 6 RELATED WORK

The problem of analyzing the performance of parallel systems has been well addressed in existing literature. This leads to speedup and efficiency, two performance metrics commonly used to evaluate the performance of parallel systems. Amdahl [3] pointed out that speedup is limited by the time needed for the serial fraction of the problem. Kumar et al. [24] and Heidelberger et al. [14] used a queueing technique to model the behavior of parallel systems. Gustafson [12] pointed out that any sufficiently large problem can be efficiently parallelized with a speedup. Eager et al. [11] pointed out that the tradeoff between speedup and efficiency can be determined by the average parallelism of the software system.

Scalability has been vigorously studied in the recent past, with an extensive amount of literature on (a) traditional computing systems such as multiprocessor systems and parallel computers [5, 15, 30, 37, 42], and (b) computing resources on public clouds [2, 10, 29, 31, 32, 34, 35, 43, 44, 47, 48]. With the assumption that there is a certain amount of computing resource (CPU, memory, disk I/O, network I/O, etc.) associated with a processor or a worker node, the primary focus of these research works is how applications can take full advantage of the computing resources offered by either multiple processors (vertical scaling) [5, 30, 42] or multiple nodes (horizontal scaling) [2, 25, 46] to achieve greater processing capacity.

Horizontal scaling has become the main technique to achieve greater processing capacity in public clouds [32, 44]. Despite the various algorithms proposed to trigger scaling actions [10, 31, 34, 35, 43, 47, 48], the underlying assumption remains the same—there is a certain amount of computing resource (CPU, memory, disk I/O, network I/O, etc.) associated with a compute instance, and a linear relationship between the amount of computing resource and the number of nodes in the worker fleet. (Whether or not the application can take full advantage of the computing resource available is another question.) This leads to the expectation that for a perfectly horizontally scalable application, its processing capacity can grow indefinitely as long as new worker nodes can be obtained from the public cloud. For horizontally scalable applications, it has been reported that their performance grows with the addition of workers nodes, although there might not be a linear relationship between performance and the number of nodes [36, 49]. However, it has not been reported that the performance of a horizontal scalable application stops growing with the addition of worker nodes that we study in this article.

The multi-tenant nature and the resulting "noisy-neighbor effect" of public clouds is well known to researchers. Numerous efforts have been carried out to understand the performance variation in public clouds. The majority of these studies were performed on Amazon EC2 [1, 4, 9, 16, 18, 38], but other public clouds are also covered extensively [13, 17, 19, 21, 26, 28, 33]. Most of these benchmark efforts are application-specific, with a heavy focus on high-performance computing, online transactional processing (OLTP) workload, and MapReduce workload. Varadarajan et al. [45] studied the possibility of virtual machine co-location on Amazon EC2, Google GCE, and Microsoft Azure. The authors demonstrate that the chances of virtual machine co-location are far higher than expected, even on public clouds that have massive datacenters with very large resource pools.

Benchmarking is usually a labor-intensive and error-prone process. Various benchmark tools have been developed to make the benchmark process easier. Li et al. [27] developed CloudCmp to compare both the performance and cost of cloud providers, using benchmark results obtained from compute service, persistent storage, and networking. Marcio et al. [41] developed CloudBench to automate benchmark tasks. In CloudBench, benchmarks were defined as high-level experiment plans, with workload templates in the medium-level and shell scripts in the low-level. This allowed CloudBench to execute complex and dynamic benchmarks with horizontal scaling. Cunha et al. [8] developed Cloud Crawler with a domain-specific language to describe the test scenario. A



RIGHTSLINK

Java-based execution engine was used to configure the test environment, execute the tests, and collect test results. Jayasinghe et al. [20] developed Expertus using secure shell and secure copy (SSH/SCP) techniques, which were capable of running the same set of jobs with the same runtime parameters on multiple nodes in parallel. Scheuner et al. [39, 40] developed Cloud WorkBench based on the notion of infrastructure-as-code. Cloud WorkBench provided support for the whole benchmark life cycle including test definition, resource provisioning and configuration, test execution, data collection, and data visualization.

All of the previous-mentioned literature adopted the performance variation approach, with the implicit assumption that more computing resource can "always" be obtained from the seemingly unlimited resource pool offered by the public cloud service provider. Apart from the implicit awareness that (a) bottlenecks exist in all systems including public clouds, and (b) the horizontal scaling technique can fail at some point, there exists no quantitative study on the limit of horizontal scaling in public clouds.

## 7 CONCLUSION

In this article, we have studied the limit of horizontal scaling in public clouds. We design and implement ScaleBench that utilizes a combination of serial and parallel evaluations to detect cloud-scale bottlenecks. We propose a CDI to quantify how likely it is for a horizontally scalable application to encounter capacity degradation when running on a public cloud at scale. We observe significant capacity degradation on three out of the four public clouds being tested. This confirms our hypothesis that there exists various cloud-scale bottlenecks in public clouds. Further, such bottlenecks can be easily detected by ScaleBench with as little as 20 instances. We perform large-scale experiments with a real-life video transcoding application on worker fleets with up to 3200 vCPU cores. We demonstrate that when the above-mentioned bottleneck is reached, the capacity of the application stops growing regardless of the growth in the number of nodes. We also demonstrate how to predict when the horizontal scaling technique can fail, based on the cloud-scale bottlenecks detected by ScaleBench. This allows public cloud users to make proper design decisions as early as possible.

## REFERENCES

[1] Sayaka Akioka and Yoichi Muraoka. 2010. HPC benchmarks on Amazon EC2. In *Proceedings of the 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. IEEE, 1029–1034.

[2] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. 2010. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th Symposium on Networked Systems Design and Implementation (NSDI)*, Vol. 10. USENIX, 89–92.

[3] Gene M. Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the American Federation of Information Processing Societies*. ACM, 483–485.

[4] Sean Kenneth Barker and Prashant Shenoy. 2010. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proceedings of the 1st Annual ACM SIGMM Conference on Multimedia Systems*. ACM, 35–46.

[5] Silas Boyd-Wickizer, Austin T. Clements, Yandong Mao, Aleksey Pesterev, M. Frans Kaashoek, Robert Morris, Nickolai Zeldovich, et al. 2010. An analysis of Linux scalability to many cores. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Vol. 10. 86–93.

[6] Michael Conley, Amin Vahdat, and George Porter. 2015. Achieving cost-efficient, data-intensive computing in the cloud. In *Proceedings of the 6th ACM Symposium on Cloud Computing*. ACM, 302–314.

[7] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 153–167.

[8] Matheus Cunha, Nabor Mendonca, and Americo Sampaio. 2013. A declarative environment for automatic performance evaluation in IaaS clouds. In *Proceedings of the 2013 IEEE 6th International Conference on Cloud Computing (CLOUD)*. IEEE, 285–292.



RIGHTSLINK

[9] Jiang Dejun, Guillaume Pierre, and Chi-Hung Chi. 2009. EC2 performance analysis for resource provisioning of service-oriented applications. In *Proceedings of the 2009 IEEE International Conference on Service-Oriented Computing (ICSOC)*. Springer, 197–207.
[10] Sourav Dutta, Sankalp Gera, Akshat Verma, and Balaji Viswanathan. 2012. Smartscale: Automatic application scaling in enterprise clouds. In *Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*. IEEE, 221–228.
[11] Derek L. Eager, John Zahorjan, and Edward D. Lazowska. 1989. Speedup versus efficiency in parallel systems. *IEEE Transactions on Computers (TC)* 38, 3 (1989), 408–423.
[12] John L. Gustafson. 1988. Reevaluating Amdahl's law. *Commun. ACM* 31, 5 (1988), 532–533.
[13] Sen He, Glenna Manns, John Saunders, Wei Wang, Lori Pollock, and Mary Lou Soffa. 2019. A statistics-based performance testing methodology for cloud applications. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 188–199.
[14] Philip Heidelberger and Kishor S. Trivedi. 1982. Queueing network models for parallel processing with asynchronous tasks. *IEEE Transactions on Computers (TC)*11 (1982), 1099–1109.
[15] Mark D. Hill. 1990. What is scalability? *ACM SIGARCH Computer Architecture News* 18, 4 (1990), 18–21.
[16] Zach Hill and Marty Humphrey. 2009. A quantitative analysis of high performance computing with Amazon's EC2 infrastructure: The death of the local cluster? In *Proceedings of the 2009 10th IEEE/ACM International Conference on Grid Computing*. IEEE, 26–33.
[17] Zach Hill, Jie Li, Ming Mao, Arkaitz Ruiz-Alvarez, and Marty Humphrey. 2010. Early observations on the performance of Windows Azure. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 367–376.
[18] Kai Hwang, Xiaoying Bai, Yue Shi, Muyang Li, Wen-Guang Chen, and Yongwei Wu. 2016. Cloud performance modeling with benchmark evaluation of elastic scaling strategies. *IEEE Transactions on Parallel and Distributed Systems* 27, 1 (2016), 130–143.
[19] Alexandru Iosup, Simon Ostermann, M. Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. 2011. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions on Parallel and Distributed Systems* 22, 6 (2011), 931–945.
[20] Deepal Jayasinghe, Josh Kimball, Siddharth Choudhary, Tao Zhu, and Calton Pu. 2013. An automated approach to create, store, and analyze large-scale experimental data in clouds. In *Proceedings of the 2013 IEEE 14th International Conference on Information Reuse and Integration (IRI)*. IEEE, 357–364.
[21] Qingye Jiang. 2012. Virtual machine performance comparison of public IaaS providers in China. In *Proceedings of the 2012 IEEE Asia Pacific Cloud Computing Congress (APCloudCC)*. IEEE, 16–19.
[22] Qingye Jiang, Young Choon Lee, and Albert Y. Zomaya. 2016. Price elasticity in the enterprise computing resource market. *IEEE Cloud Computing* 3, 1 (2016), 24–31.
[23] Qingye Jiang, Young Choon Lee, and Albert Y. Zomaya. 2019. Scalable video transcoding in public clouds. In *Proceedings of the 2019 19th IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing (CCGrid)*. IEEE, 70–75.
[24] B. Kumar and Timothy A. Gonsalves. 1979. Modelling and analysis of distributed software systems. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles*. ACM, 2–8.
[25] Avinash Lakshman and Prashant Malik. 2010. Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review* 44, 2 (2010), 35–40.
[26] Philipp Leitner and Jürgen Cito. 2016. Patterns in the Chaos– A study of performance variation and predictability in public IaaS clouds. *ACM Transactions on Internet Technology (TOIT)* 16, 3 (2016), 15.
[27] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. 2010. CloudCmp: Comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. ACM, 1–14.
[28] Zheng Li, Liam OBrien, Rajiv Ranjan, and Miranda Zhang. 2013. Early observations on performance of Google compute engine for scientific computing. In *Proceedings of the 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, Vol. 1. IEEE, 1–8.
[29] Harold C. Lim, Shivnath Babu, Jeffrey S. Chase, and Sujay S. Parekh. 2009. Automated control in cloud computing: Challenges and opportunities. In *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds*. ACM, 13–18.
[30] Benjamin Linder. 1993. Oracle parallel RDBMS on massively parallel systems. In *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*. IEEE, 67–68.
[31] Tania Lorido-Botrán, José Miguel-Alonso, and Jose Antonio Lozano. 2012. Auto-scaling techniques for elastic applications in cloud environments. *Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09* 12 (2012), 2012.



RIGHTSLINK

[32] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A. Lozano. 2014. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing* 12, 4 (2014), 559–592.
[33] Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, and Robert Ricci. 2018. Taming performance variability. In *Proceedings of the 2018 13th {USENIX} Symposium on Operating Systems Design and Implementation (OSDI'18)*. 409–425.
[34] Athanasios Naskos, Emmanouela Stachtiari, Anastasios Gounaris, Panagiotis Katsaros, Dimitrios Tsoumakos, Ioannis Konstantinou, and Spyros Sioutas. 2015. Dependable horizontal scaling based on probabilistic model checking. In *Proceedings of the 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 31–40.
[35] Ashkan Paya and Dan C. Marinescu. 2017. Energy-aware load balancing and application scaling for the cloud ecosystem. *IEEE Transactions on Cloud Computing* 5, 1 (2017), 15–27.
[36] George Porter. 2010. Decoupling storage and computation in Hadoop with super data nodes. *ACM SIGOPS Operating Systems Review* 44, 2 (2010), 41–46.
[37] Dan Pritchett. 2008. Base: An ACID alternative. *Queue* 6, 3 (2008), 48–55.
[38] Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. 2010. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proceedings of the VLDB Endowment* 3, 1–2 (2010), 460–471.
[39] Joel Scheuner, Jürgen Cito, Philipp Leitner, and Harald Gall. 2015. Cloud workbench: Benchmarking IaaS providers based on infrastructure-as-code. In *Proceedings of the 24th International Conference on World Wide Web (WWW)*. ACM, 239–242.
[40] Joel Scheuner, Philipp Leitner, Jürgen Cito, and Harald Gall. 2014. Cloud work bench–infrastructure-as-code based cloud benchmarking. In *Proceedings of the 2014 IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 246–253.
[41] Marcio Silva, Michael R. Hines, Diego Gallo, Qi Liu, Kyung Dong Ryu, and Dilma Da Silva. 2013. Cloudbench: Experiment automation for cloud environments. In *Proceedings of the 2013 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 302–311.
[42] Xian-He Sun and Diane T. Rover. 1994. Scalability of parallel algorithm-machine combinations. *IEEE Transactions on Parallel and Distributed Systems* 5, 6 (1994), 599–613.
[43] Hien Nguyen Van, Frédéric Dang Tran, and Jean-Marc Menaud. 2010. Performance and power management for cloud infrastructures. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD)*. IEEE, 329–336.
[44] Luis M Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. 2011. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review* 41, 1 (2011), 45–52.
[45] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2015. A placement vulnerability study in multi-tenant public clouds. In *Proceedings of the 24th {USENIX} Security Symposium ({USENIX} Security 15)*. 913–928.
[46] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. 2006. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, 307–320.
[47] Jingqi Yang, Chuanchang Liu, Yanlei Shang, Bo Cheng, Zexiang Mao, Chunhong Liu, Lisha Niu, and Junliang Chen. 2014. A cost-aware auto-scaling approach using the workload prediction in service clouds. *Information Systems Frontiers* 16, 1 (2014), 7–18.
[48] Jingqi Yang, Chuanchang Liu, Yanlei Shang, Zexiang Mao, and Junliang Chen. 2013. Workload predicting-based automatic scaling in service clouds. In *Proceedings of the 2013 IEEE 6th International Conference on Cloud Computing (CLOUD)*. IEEE, 810–815.
[49] Dongfang Zhao, Xu Yang, Iman Sadooghi, Gabriele Garzoglio, Steven Timm, and Ioan Raicu. 2015. High-performance storage support for scientific applications on the cloud. In *Proceedings of the 6th Workshop on Scientific Cloud Computing*. ACM, 33–36.

Received August 2019; revised November 2019; accepted November 2019



RIGHTSLINK