Benchmarking in the Cloud: What it Should, Can, and Cannot Be

Enno Folkerts[1], Alexander Alexandrov[2], Kai Sachs[1],
Alexandru Iosup[3], Volker Markl[2], and Cafer Tosun[1]

[1] SAP AG, 69190 Walldorf, Germany firstname.lastname@sap.com
[2] TU Berlin, Germany firstname.lastname@tu-berlin.de
[3] Delft University of Technology, The Netherlands A.Iosup@tudelft.nl

With the increasing adoption of Cloud Computing, we observe an increasing need for Cloud Benchmarks, in order to assess the performance of Cloud infrastructures and software stacks, to assist with provisioning decisions for Cloud users, and to compare Cloud offerings. We understand our paper as one of the first systematic approaches to the topic of Cloud Benchmarks. Our driving principle is that Cloud Benchmarks must consider end-to-end performance and pricing, taking into account that services are delivered over the Internet. This requirement yields new challenges for benchmarking and requires us to revisit existing benchmarking practices in order to adopt them to the Cloud.

1 Introduction

Creating good benchmarks has been considered a "dark art" for a long time because of the many subtleties that ultimately may influence the adoption (and consequently the success) of a benchmark. Nevertheless, the body of related research work suggests a number of widely accepted guidelines and quality criteria which have to be considered in the design and execution of computer system benchmarks.

In this paper, we seek to extent these quality criteria for benchmarking in the Cloud. For the purposes of our discussion, by *benchmarking in the Cloud* we mean the use of Cloud services in the respective (distributed) system under test (SUT). We believe that *building* the benchmark is only half of the story and *execution (operation)* deserves at least as much attention, especially in the discussed complex distributed systems context.

Our work is mainly inspired by *The art of building a good benchmark* [1] by Karl Huppler and *How is the Weather tomorrow? Towards a Benchmark for the Cloud* [2] by Carsten Binnig, Donald Kossmann, Tim Kraska and Simon Loesing. The Dagstuhl Seminar on Information Management in the Cloud held in August 2011 was the starting point for the actual work on the paper. We would like to thank Helmut Krcmar and André Bögelsack who pointed out the business aspect of the topic, as well as Nick Lanham and Dean Jacobs for their suggestions and feedback.

The paper is structured as follows. Section 2 gives an overview of benchmarking in general. Section 3 introduces the topic of benchmarking in the Cloud. In

Section 4 and Section 5 we present use cases in the Cloud and go through the necessary steps for building respective benchmarks. Section 6 highlights the challenges for building a good benchmark in the Cloud and proposes first solutions. Finally, Section 7 concludes and presents ideas for future research.

2 Benchmarking in a Nutshell

This section gives a brief overview on the topic of benchmarking. We will discuss the objectives of benchmarks, see how benchmarks operate, and then will elaborate how benchmarks try to meet their objectives.

2.1 What is the task of a benchmark?

Benchmarks are tools for answering the common question *"Which is the best system in a given domain?"*. For example, the SPECCpu benchmark [3] answers the question *"Which is the best CPU?"*, and the TPC-C benchmark [4] answers the question *"Which is the best database system for OLTP?"*.

The concrete interpretation of *"best"* depends on the benchmarking objective and is the first question that has to be answered when designing a new benchmark. As a systematic approach for answering this question, Florescu and Kossmann suggest to look at the properties and constraints of the systems to be benchmarked [5]. The number one property has to be optimized while lower priority properties give rise to constraints. A benchmark therefore can be seen as a way to specify these priorities and constraints in a well-defined manner. The task of the benchmark then is to report how well different systems perform with respect to the optimized priority under the given constraints.

In practice, benchmarks are used to assist decisions about the most economical provisioning strategy as well as to gain insights about performance bottlenecks.

2.2 How do benchmarks do their task?

Benchmarks pick a representative scenario for the given domain. They define rules how to setup and run the scenario, and how to obtain measurement results.

Benchmark definitions often refer to the concept of a *System Under Test (SUT)*. The SUT is a collection of components necessary to run the benchmark scenario. The idea of a SUT is to define a complete application architecture containing one or more *components of interest*. In a typical SUT, however, not all components are of principal interest for the benchmark. We refer to the remaining SUT components as *purely functional components*.

Benchmarks measure the the behaviour of a complete SUT. In order to isolate information about the component of interest, complete knowledge about all components involved is essential. That is why all SUT components are subject to strict run and disclosure rules. Benchmark components initiating the workload are called *drivers*, and are not part of the SUT.

2.3 Benchmark Requirements

A benchmark is considered to be good if all stakeholders believe that it provides true and meaningful results. There are a number of publications that try to provide guidelines on the subject of benchmark design and implementation. Almost all of them are based on Gray's seminal work [6]. Huppler recently provided a good survey on different benchmarking criteria in [1]. Workload requirements are investigated in [7–10]. Based on this previous work, we define the following three groups of requirements:

1. *General Requirements* – this group contains generic requirements.
   (a) Strong Target Audience – the target audience must be of considerable size and interested to obtain the information.
   (b) Relevant – the benchmark results have to measure the performance of the typical operation within the problem domain.
   (c) Economical – the cost of running the benchmark should be affordable.
   (d) Simple – understandable benchmarks create trust.
2. *Implementation Requirements* – this group contains requirements regarding implementation and technical challenges.
   (a) Fair and Portable – all compared systems can participate equally.
   (b) Repeatable – the benchmark results can be reproduced by rerunning the benchmark under similar conditions with the same result.
   (c) Realistic and Comprehensive – the workload exercises all SUT features typically used in the major classes of target applications.
   (d) Configurable – a benchmark should provide a flexible performance analysis framework allowing users to configure and customize the workload.
3. *Workload Requirements* – contains requirements regarding the workload definition and its interactions.
   (a) Representativeness – the benchmark should be based on a workload scenario that contains a representative set of interactions.
   (b) Scalable – Scalability should be supported in a manner that preserves the relation to the real-life business scenario modeled.
   (c) Metric – a meaningful and understandable metric is required to report about the SUT reactions to the load.

In the following sections, we evaluate our results using these requirements and discuss how they can be fulfilled in our scenarios.

3 Benchmarking in the Cloud

For a definition of Cloud Computing, we refer to the *NIST Definition of Cloud Computing* [11]. This definition comes with three different service models: *Infrastructure as a Service (IaaS)*, *Platform as a Service (PaaS)*, and *Software as a service (SaaS)*. These service models are commonly referred to as *Service Levels* and can be understood as different levels of software abstraction. These layers do not define a fixed homogeneous set – authors and businesses often introduce new

Something as a Service (XaaS) terminologies. Due to space limitations, here we only mention Youseff et al. [12], who extend the three layer model with *Hardware as a Service*, *Data as a Service*, and *Communication as a Service* concepts.

Under *Benchmarking in the Cloud*, we understand the process of benchmarking services provided by the Cloud. A *Cloud Benchmark* therefore for us is a benchmark in which the SUT contains a Cloud service as component of interest. A good summary of this topic was provided in [13].

3.1 Cloud Actors

To introduce the different Cloud actors, we take a business orientated view of Cloud Computing going beyond the simple consumer/provider model. We claim that each service layer brings its own actors who add value to the level below. Different types of actors might have different benchmark requirements for the same SUT. To have a strong target audience, a benchmark has to address the appropriate actors.

Leimeister et al. [14] argue that the actors in the Cloud form a business value network rather than a traditional business value chain. We identify the following actors in a Cloud-centric business value network (Figure 1): *IT Vendors* develop infrastructure software and operate infrastructure services; *Service Providers* develop and operate services; *Service Aggregators* offer new services by combining preexisting services; *Service Platform Providers* offer an environment for developing Cloud applications; *Consulting* supports customers with selecting and implementing Cloud services; *Customers* are the end-users of Cloud services.

Note that [14] uses the term *Infrastructure Provider* for what we call *IT Vendor*. We deviate from [14] to stress the fact that vendors that offer software that enables Cloud services should also be considered part of this actor group. We also use the term *Customer* where others might use the term *Consumer*. We decided to adhere to [14] in this case because service aggregators and service platform providers are consumers just as customers are.

3.2 The System Under Test

The above definition of a cloud benchmark requires that at least one component of interest within the benchmark SUT is a cloud service. This leads to several implications, which we now briefly discuss.

SUT Disclosure. A common benchmark requirement is to lay open all properties of the involved SUT components. This requirement is no longer realistic when the SUT contains public cloud services. We therefore propose to consider the following relaxation: *All properties of SUT components visible to their clients should be laid open.* In terms of white-box vs. black-box benchmarking this leads to nearly white-box IaaS benchmarks, grey-box PaaS benchmarks and black-box SaaS benchmarks. Overcoming the SUT disclosure issue was also recently discussed by the SPEC OSG Cloud Computing Workgroup. Their White Paper [15] provides further information.

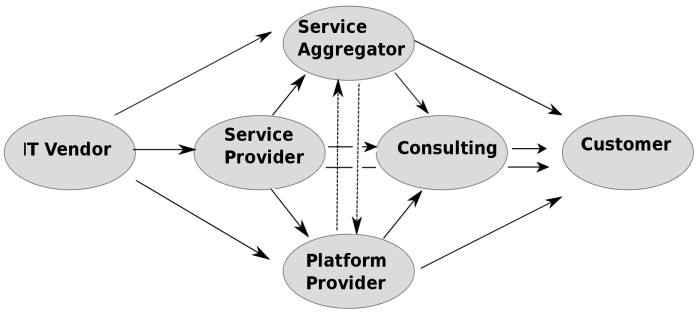

Fig. 1. Cloud Actors and their Value Network

SUT Uniqueness. Traditional benchmarks run on SUT components copied into an isolated test environment. In a cloud environment, components are modeled as services. These are single instance entities, cannot be copied, and a clear isolation is not possible. During a benchmark run the SUT services most likely will be shared with third party clients, and avoiding external clients is neither possible nor desired.

Carving out a SUT. One possible approach to deal with the disclosure and uniqueness issues discussed above is to carve out a dedicated set of servers within a public cloud and have these deliver the services for the SUT. We believe that this will not lead to relevant results.

4 Use Cases

In this section we present a list of sample use cases. The list does not intend to be complete, but rather should help us illustrate the layers and actors defined above and motivate the discussion of different optimization questions. In Section 5 we show how appropriate Cloud benchmarks help answer these questions, and in Section 6 we identify and discuss the most important challenges that these benchmarks should resolve.

4.1 Simple IaaS

Businesses use Cloud IaaS for outsourcing of non-critical processes like testing, training and reference landscapes. They buy IaaS resources and install the desired systems, but do not use them all the time. The expected load is moderate to low, being created mainly by non-critical offline processes. Our experience with such scenarios shows systems with 100GB disk space, 32GB memory, and 4

CPU cores. The average load of a running system uses about 10% CPU resources and 25% of memory or disk space.

The IaaS resources are not expected to scale with regard to the load per system. Scalability is expected with regards to the number of systems that can be started and shut down.

4.2 Online Gaming Platform

GameX is produced by *ACME Games*.

ACME Games runs its own server clusters which should host on average hundreds of game and mini-game sessions, each with varying number of users (say between 0 and 5,000 for the core game, and from 5 to 50 for the mini- or smaller game instances [16]), and length ranging of 20-30 mins to several hours.

When players overload the servers, no new player sessions can be established. Waiting to login is one of the worst breaches of the expected Service Level Agreements for *GameX*, so *ACME Games* has to avoid these situations at all cost. Because the resource consumption of interaction operations between players does not always scale linearly with the number of players, and, in fact, in some cases may grow cubically in the number of players, *ACME Games* has to be either overly-conservative in its server capacity estimates or risk severe overloads in some circumstances [17].

Instead of over-provisioning, which as seen is highly undesirable, *ACME Games* can use Cloud technology. They can lease machines on-demand to alleviate sudden surges in their workload. This solution needs to fulfill strict Service Level Agreements in terms of response time, which includes a non-trivial network latency component, and computing workload distribution and interference. For its new games, *ACME Games* may also lease (reserve) instances from a Cloud until the game is properly measured and future workloads can be predicted.

4.3 High Workload Analytical Platform

ACME Analytics is a start-up that wants to provide Big Data analytics services. Their target customers produce large amounts of data and want to analyze it on a daily or weekly basis. Potential customers may be smart grid or mobile phone providers. The company expects up to 500 TB data per client and as much as 1 PB of data shared among clients. An average task is expected to use 50TB of the private and 1 TB of the shared data. Clearly, using a massively-parallel data processing platform, such as MapReduce, as an off-the-shelf Cloud service is the most lucrative technical solution for *ACME Analytics* because of the low upfront investment and maintenance cost. The first technical decision that *ACME Analytics* has to make therefore is which PaaS to use.

5 Building Benchmarks in the Cloud

In this section we go through the previous use cases and try to envision how benchmarks in the respective area could be build.

5.1 Simple IaaS

First, we need to refine the optimization question for the use case. The consumer needs many parallel instances of some application for their test or training systems. There is no necessity to scale up a single application instance. It is expected to save cost by shutting down application instances. We can therefore recap the optimization question posed by the use case as follows: *Which IaaS does most effectively host a bunch of parallel mid-size application instances? This includes reducing cost when fewer application instances are active.*

Next, we discuss how the SUT and the workload should look like. We are not interested in a distributed or cluster scenario, so we can adapt a well known workload, say TPC-C, and run it independently on all instances.

To adapt TPC-C, we first pick a `tpmC` value representative for a mid-size load. In addition, we also set the maximum number of independent application instances that will participate for a single benchmark run. Let us put `maxInst`=250.

We will consider two workload variants: (1) running all instances with the same `tpmC`, and (2) running a varying amount of instances over the benchmark runtime. Workload (1) is intended to measure how many IaaS nodes are required to make `maxInst` instances pass the TPC-C SLA requirements. Because different providers are expected to have different nodes, a final comparison of the respective services will only be possible by comparing the price of the services consumed. Workload (2) measures the elasticity of the underlying IaaS by comparing the price of the full workload with the price of the varying workload. For the second workload, we propose a scheduling mechanism with number of active instances defined by the formula:

$$actInstances(\texttt{timeElapsed}) = \left| \frac{\texttt{maxInst}}{2} \times \left(1 - \cos\left(\frac{3\pi \times \texttt{timeElapsed}}{\texttt{totalRuntime}}\right)\right)\right|$$

These are only a few ideas about a parallel load IaaS benchmark. We list further design questions for such a benchmark.

1. As we are not benchmarking the database we can fix the database system to be used. But this would violate the fairness and portability requirement.
2. Is it allowed to use different schemata of a database instance for different TPC-C instances?
3. Is it allowed to migrate TPC-C instances between server nodes during a benchmark run?
4. Should we rather not design for possible multiple TPC-C per node and scale by increasing the `tpmC` until a node is fully loaded?
5. Where should the Remote Terminal Emulator (the driver) be located?

We discuss these questions in Section 6.

5.2 Online Gaming Platform

There currently exists no online gaming benchmark. Relevant prior work on the prerequisites of designing online gaming benchmarks exists, either in the form

of game workload characterizations or of benchmarks built for other types of media. The voluminous body of work on game workload characterization has been recently surveyed [18]. For related benchmarks, we refer to ALPBench [19], MediaBench [20], and MiBench [21].

We hold that the Gaming use case rather gives rise to parallel than a distributed scenario. Still, it is much more complex than the Simple IaaS use case. In this case the various types of system operations necessary to fulfill various requests may lead to widely different response times. Another fundamental difference is that requests need to be fulfilled in different amount of times before the users consider their mental Service Level Agreement breached and decide to move to another game. For example, role-playing requests may be fulfilled within 1 second [22, 23]. For strategic decisions the response time needs to be around 300 milliseconds [24], and requests for first-person activities need to be fulfilled in under 100 milliseconds [23], [25].

Other important metrics for ACME Games are the 99th percentile of the wait time distribution and the the 99th percentile of the distribution of fraction of dropped requests. These high limits (vs the traditional 95th percentile) stem from the way players join and leave games as the result of positive and negative trends, respectively. Strong community ties between players [26], [27] indicate that a percentage as low as 1% of unhappy players may trigger the departure of 10-20% in a matter of days [17], e.g. via social media rants, in-game negative adverts, and plain group-based discussions.

5.3 High Workload Analytical Platform

In this case, *ACME Consulting* acts as service aggregator and has two options. They may either bundle services of an existing analytical platform or deploy their own analytical tools on an existing infrastructure service. To compare the resulting service they need a benchmark build around a set of representative analytical tasks. Most research in the area in done on actual MapReduce benchmarks like MRBench [28] or designing appropriate MapReduce workloads [29]. Pavlo et al. [30] show how to have analytical workload run by both MapReduce and Distributed Databases and compare the results. These approaches make considerable progress defining representative workloads. They do not deal with services and do not intend to define Cloud benchmarks. Here are a few ideas how an analytical Cloud benchmark could look like:

1. Start with a set of analytical tasks from the latest references.
2. For each task find an appropriate runtime SLA defining a 90% percentile.
3. Run the tasks concurrently. Run the whole workload several times.
4. Scale the tasks with the amount of data analyzed. Keep things simple by scaling up all task data sizes with the same linear factor `anScale`.
5. We expect the SUT to acquire more server nodes as the amount of data to be analyzed increases. The primary metric is the maximal `anScale`, that can be reached.

6. Devise a maximal scaling factor `maxAnScaleYYYY`. The benchmark should not scale the load further than `maxAnScaleYYYY`.
7. Also report the price of the services used so that services can be compared if they reach the maximal `anScale`.
8. Periodically (for example once per year) revise and agree on a new (larger) `maxAnScaleYYYY`.
9. Similarly to the Simple IaaS use case report elasticity by running with a varying load.

In the next section we discusses the challenges for the hypothetical benchmarks presented above in more detail.

6 The Art of Building and Running a Good Benchmark in the Cloud

We believe, that there are four main steps for building and running a good benchmark in the Cloud. These are *Meaningful Metric*, *Workload Design*, *Workload Implementation* and *Creating Trust*. In this section we discuss the inherent challenges in these steps.

6.1 Meaningful Metric

A metric is a function that transforms measured results into a form that is easily understood by the system analyst. The most simple metric is the runtime metric, which reports either median, average, maximum or even minimum runtime among transactions run by the SUT. When systems have to deal with concurrent transactions, it makes more sense to consider a throughput metric reporting the maximal number of transactions adhering to some runtime SLA. Some benchmarks (mainly those designed to support business decisions) also report the cost of the underlying system or the cost of running a single transaction at maximal load. There is an ongoing debate if reporting cost makes sense [31]. Kossmann et al. have devoted a sequence of papers to this topic in its Cloud context [5, 2, 32], arguing that in this case cost should be the primary metric. The argument is motivated by the fact that in theory Cloud technology should provide infinite scalability, which makes the classic throughput metric obsolete. However, in [32] the authors observe that (to no surprise) infinite scalability is an illusion and for most providers breaks down sooner or later due to bottlenecks in the SUT architecture. In that case it makes perfect sense to report how far the load can be scaled up.

Challenge 1: Price vs. Performance Metric. For the parallel load of the Simple IaaS use case we had decided to report the number of nodes required to run the 250 TPC-C instances. As nodes of different vendors will most probably have different characteristics, the number of nodes is in fact not a suitable metric. Money is the only possible yardstick and therefore we have to report the price

for these nodes. We propose to use the Price Metric as primary metric when dealing with parallel load: *Report the minimal cost to run* `maxInst` *of a given application and a given workload on the SUT.*
For a distributed load we propose to use a mixed Price/Performance Metric as suggested by the High Analytical Workload Platform use case: *Scale up the load along a well defined dimension, but do not pass a very ambitious limit of* `maxScaleYYYY` *which should be reconsidered annually. Also report the price of the SUT. This enables comparison in case systems reach* `maxScaleYYYY`*.*

Challenge 2: Elasticity Metric. We propose to use a Elasticity Metric as secondary metric. How can elasticity of a service under test be measured? Previous work has introduced for this purpose concepts such as over- and under-provisioning of resources [17, 33], or counted the number of SLA breaches [17] during periods of elastic system activity. However, measuring and characterizing elasticity remain activities without theoretical support. Even understanding which periods are elastic, that is, distinguishing between normal fluctuations and significant elastic changes in the system behavior, requires advances in the current body of knowledge. Moreover, the proposal in [17, 33] relies on the benchmark being able to collect CPU consumption information of the provider system. In some cases these numbers might not be freely available to the consumer. Consequently a consumer benchmark should not rely on these numbers. Binnig et al. [2] define a Scalability Metric, which does not take into account CPU info and solely relies on the successful interactions under an increasing load. This methodology could be extended to also capture elasticity. Nevertheless, it is of little interest to the consumer if the provider can deal effectively with a varying load. What the consumer needs to know is whether a varied and reduced load leads to a reduced bill. We therefore propose to measure elasticity by running a varying workload and compare the resulting price with the price for the full load. Some details can be found in the discussion of the Simple IaaS benchmark.

Challenge 3: Percentile. Which percentiles are appropriate for the response time SLAs of Cloud like interactions? One might argue, that the Cloud calls for higher percentiles. Disappointed users might just move to the next service offered. We hold that percentiles depend on the respective scenario. For the Simple IaaS scenario the 90% percentiles of TPC-C are fine. For the Gaming scenario they are not. We propose not to run the same benchmark (scenario) with different percentiles but to have different benchmarks modeling different scenarios and respective percentiles.

Discussing the metric topic in the Cloud community we were asked to also consider several other metrics like consistency, security, user experience and trust. In our opinion each scenario has its own consistency requirement. The consistency level of a service has to fulfill this consistency requirement. If a service promises a higher degree of consistency than required, this should be stated

in the benchmark report. The same more or less holds for security. User experience and trust are hard to capture. We intend to create trust by doing the benchmarking right.

6.2 Workload Design

The workload of a benchmark must be designed towards the metric to be used. It has to model real world workloads on a given application scenario. Workload design is a traditional challenge in the design of benchmarks.

Challenge 4: Limit the Resources Acquired. Since many Clouds try to provide the illusion of infinite scale, benchmarks cannot merely load the system until it breaks down. This applies in particular for parallel load, where we expect to be able to crash the complete service. This is not realistic and might violate the benchmark requirement to stay within economical boundaries. We have to devise a maximal scaling factor limiting the load. However, having this accepted by the majority of stakeholders poses a serious challenge. We propose to annually renew the maximal scaling factor. We expect a discussion about having a maximal scaling factor or not. Those against could argue, that providers have to take care not to have their services crashed. A benchmark would check implicitly, if this kind of check is in place.

Challenge 5: Variable Load. As user-triggered elasticity and automatic adaptivity are expected features of Cloud services, benchmarks can no longer focus solely on steady-state workloads. For the Simple IaaS use case we proposed to use a harmonic variability of the load. In general, capturing and using characteristics of real Cloud workloads might lead to better accepted benchmarks.

Challenge 6: Scalability. The benchmark requirements listed in Section 2.3 ask for Scalability. This means that the benchmark has to be enabled to have the load against the SUT increased along a well defined scale. What is the best candidate for a 'well defined scale' for our use case? The answer to this question is not obvious.

Let us return to the Simple IaaS use case and recap the design of its benchmark. The general question is: how well can a service host multiple instances of a given application? To answer this question the benchmark increases the load and reports how far it can get. We have several options to increase the load:

1. Increase the number of users per application instance.
2. Increase the size of the application instance.
3. Increase the number of application instances.
4. Increase the number of application instance per node.

As we do not model for high load per application instance the first two options are not relevant. The third option leaves open the question where to set

the limit discussed in the previous challenge. We choose the last option because it addresses resource sharing. Once we deal with distributed load we face different options. In the case of MapReduce scenarios we choose the size of the data to be analysed as 'well defined scale'.

6.3 Workload Implementation

Challenge 7: Workload Generation. The increased complexity of the workloads also imposes challenges for the implementation of workload generator programs. First, efficient scalability is a hard requirement because of the expected workload sizes. Second, the workload generators should be able to implement all relevant characteristics of the designed application scenario.

As an example, consider the analytical platform use-case from Section 5.3. A relevant benchmark should use test data with similar order of magnitude, so a natural problem that becomes evident here is how to ship benchmark data of that magnitude to the SUT. Clearly, importing data from remote locations or using sequential data generators are not feasible options due to the unrealistic amount of required time (according to our estimates generating 1PB of TPC-H data for instance will take at least a month). The most promising alternative to solve this problem is to utilize the computational resources in the Cloud environment and generate the data and the workloads in a highly parallel manner. Recent work in the area of scalable data generation [34, 35] demonstrates how a special class of pseudo-random number generators (PRNGs) can be exploited for efficient generation of complex, update-aware data and workloads in a highly parallel manner. In general, partitionable PRNGs can be used to ensure repeatability and protect against unwanted correlations for all sorts of parallel simulations.

Challenge 8: Fairness and Portability. Like Binnig et al. [2] we propose to have Cloud benchmarks model the complete application stack. In case we do not benchmark SaaS services, some party has to provide an implementation of the respective application to be deployed on the service under test. What kind of rules and restrictions should apply for the implementation of this application? We have to take care not to violate the Fairness and Portability requirements. In the Simple IaaS use case we discussed the option to fix a database system for the benchmark. In our opinion, this would violate the Fairness and Portability requirements. Allowing different database systems increases the degrees of freedom. This in turn makes it hard to compare the underlying IaaS SUT. How can this challenge be overcome? Here is our solution:

1. Set application specs without touching implementation details.
2. Service providers are free to provide their own optimal implementation of the respective application.
3. Service providers can require submissions to be based on their own implementation.

With this solution services allowing for good implementations will have an advantage, but this is only fair.

PaaS providers offer a zoo of different languages, data stores and application servers. Some of these follow standards, some do not and others are heading to become the de facto standard. PaaS benchmarks have the choice either to be generic enough to cover all or most of these offers or to restrict themselves to a well defined standard, which is implemented by a considerable group of service providers. We propose to handle the first option like the IaaS case discussed above. This then leads to a single benchmark, which can be used for IaaS and PaaS services. This is the approach of Kossmann et al. [32], who implement the TPC-W benchmark for each of the services under test. The second PaaS option might lead to a reuse of existing benchmarks like SPECjEnterprise in the Cloud.

We conclude the discussion with a short remark about SaaS benchmarks. We cannot expect to have one benchmark for all SaaS offerings. Still, benchmarks for an important application domain like Human Capital Management(HCM) are worth consideration. Their biggest challenge would be to define a set of representative transactions, which most HCM providers offer. The more specialized the services under test are, the more difficult it is to find a common denominator.

6.4 Creating Trust

Trust, which is a major concern for any Cloud operation [36], is particularly important for benchmarking Cloud services. Choosing a representative benchmark scenario and properly dealing with above design and implementation challenges supports creating trust. We hold that we must also consider benchmark operations and list three operational challenges.

Challenge 9: Location. Benchmarking in the Cloud raises a non-trivial challenge in deciding where each component of the benchmark is located. Should the Cloud provider host the driver? Should the request queues of the user [37] be located close to the Cloud or even in different time zones? Does this decision depend on the workload or not? If yes, is there a general rule of thumb that can help us decide where to place the driver?

Challenge 10: Ownership. Which actor, from the group introduced in Section 3.1, should run the benchmark? How to prevent that the Cloud service provider "games" the results, for example by putting more resources into the benchmarked service? Should the Cloud service provider be informed about when the benchmark is being run in their system?

Challenge 11: Repeatability. We expect a variability in the results reported by a benchmark and list three possible reasons. a) The performance variability

of production IaaS Clouds has been investigated [38] and found to exhibit pronounced time patterns at various time scales. The main reason is that Cloud services time-share and may over-commit their resources. b) Ever-changing customer load may affect the performance of Cloud infrastructures. c) Moreover, providers are liable to change prices, which directly affects the proposed PricePerformance metrics. Thus, benchmarking results may vary significantly over time. In contrast to traditional benchmarking of large-scale computing systems, what is the value of numbers measured at any particular point in time? How to ensure the repeatability of results? Should benchmarks be simply re-run periodically, therefore lowering the economical requirement, or should the repeatability requirement be lowered or even not imposed?

Our first draft of a solution to these operational challenges is to set up an independent consortium. The main tasks of this consortium would be to:

1. Provide a geo-diverse driver Cloud.
2. Run benchmarks without further notice to the provider.
3. Rerun benchmarks periodically.
4. Charge benchmark runs to the provider.
5. Offer different levels of trust by having runs repeated more or less frequently.
6. Store benchmark applications implemented (see Challenge 8) by the provider or third party.

7 Conclusion

Benchmarking plays an important role in the wide-spread adoption of cloud computing technologies. General expectations of ubiquitous, uninterrupted, on-demand, and elastic cloud services must be met through innovative yet universally accepted benchmarking practices. In this work we described our understanding what benchmarking should, can, and cannot be. This understanding is governed by general benchmark requirements listed in Section 2.3 . It is also based on a sequence of papers [5], [2], [32] by Kossmann et al. and the respective experiments performed.

We first defined the actors involved in cloud benchmarking, including their value network, and the system under test (SUT). Unlike traditional benchmarking, the SUT includes numerous components that are either black boxes or inherently unstable. Next, we analyzed several use cases where benchmarking can play a significant role, and discussed the main challenges in building scenario-specific benchmarks. Last, we collected the challenges of scenario-specific benchmarks and proposed initial steps towards their solution. Besides proposing solutions for technical challenges we propose founding a consortium, which is able to tackle the operational challenges. We hope to be able to discuss our solutions with the TPC audience and are strongly committed to use our current presence in the related SPEC working groups to foster the adoption of these benchmarking technologies.

References

1. Huppler, K.: The art of building a good benchmark. In Nambiar, R.O., Poess, M., eds.: TPCTC. Volume 5895 of Lecture Notes in Computer Science., Springer (2009) 18–30
2. Binnig, C., Kossmann, D., Kraska, T., Loesing, S.: How is the weather tomorrow?: towards a benchmark for the cloud. In: DBTest, ACM (2009)
3. SPEC: The SPEC CPU2006 Benchmark URL: http://www.spec.org/cpu2006/.
4. TPC: The TPC-C Benchmark URL: http://www.tpc.org/tpcc/.
5. Florescu, D., Kossmann, D.: Rethinking cost and performance of database systems. SIGMOD Record **38**(1) (2009) 43–48
6. Gray, J., ed.: The Benchmark Handbook for Database and Transaction Systems (2nd Edition). Morgan Kaufmann (1993)
7. Kounev, S.: Performance Engineering of Distributed Component-Based Systems - Benchmarking, Modeling and Performance Prediction. PhD thesis, Technische Universität Darmstadt (2005)
8. Sachs, K., Kounev, S., Bacon, J., Buchmann, A.: Performance evaluation of message-oriented middleware using the SPECjms2007 benchmark. Performance Evaluation **66**(8) (Aug 2009) 410–434
9. Sachs, K.: Performance Modeling and Benchmarking of Event-Based Systems. PhD thesis, TU Darmstadt (2011)
10. Madeira, H., Vieira, M., Sachs, K., Kounev, S. Dagstuhl Seminar 10292. In: Resilience Benchmarking. Springer Verlag (2011)
11. NIST: *The NIST Definition of Cloud Computing* (2011) http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.
12. Youseff, L., Butrico, M., Silva, D.D.: Towards a unified ontology of cloud computing. In: Proc.of the Grid Computing Environments Workshop(GCE08). (2008)
13. Huppler, K.: Benchmarking with your head in the cloud. TPCTC (2011)
14. Leimeister, S., Böhm, M., Riedl, C., Krcmar, H.: The business perspective of cloud computing: Actors, roles and value networks. In Alexander, P.M., Turpin, M., van Deventer, J.P., eds.: ECIS. (2010)
15. SPEC Open Systems Group: Report on cloud computing to the OSG Steering Committee. Technical Report OSG-wg-final-20120214 (Feb 2012)
16. Shen, S., Visser, O., Iosup, A.: Rtsenv: An experimental environment for real-time strategy games. In Shirmohammadi, S., Griwodz, C., eds.: NETGAMES, IEEE (2011) 1–6
17. Nae, V., Iosup, A., Prodan, R.: Dynamic resource provisioning in massively multiplayer online games. IEEE Trans. Parallel Distrib. Syst. **22**(3) (2011) 380–395
18. Ratti, S., Hariri, B., Shirmohammadi, S.: A survey of first-person shooter gaming traffic on the internet. IEEE Internet Computing **14**(5) (2010) 60–69
19. Li, M., Sasanka, R., Adve, S., Chen, Y., Debes, E.: The ALPBench benchmark suite for complex multimedia applications. In: Workload Characterization Symposium, 2005. Proceedings of the IEEE International. (2005) 34–45
20. Lee, C., Potkonjak, M., Mangione-Smith, W.H.: Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In: MICRO. (1997) 330–335
21. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: MiBench: A free, commercially representative embedded benchmark suite. In: Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538), IEEE (2001) 3–14

22. Fritsch, T., Ritter, H., Schiller, J.H.: The effect of latency and network limitations on mmorpgs: a field study of everquest2. In: NETGAMES, ACM (2005) 1–9
23. Chen, K.T., Huang, P., Lei, C.L.: How sensitive are online gamers to network quality? Commun. ACM **49**(11) (2006) 34–38
24. Claypool, M.: The effect of latency on user performance in real-time strategy games. Computer Networks **49**(1) (2005) 52–70
25. Beigbeder, T., Coughlan, R., Lusher, C., Plunkett, J., Agu, E., Claypool, M.: The effects of loss and latency on user performance in unreal tournament 2003. In chang Feng, W., ed.: NETGAMES, ACM (2004) 144–151
26. Balint, M., Posea, V., Dimitriu, A., Iosup, A.: User behavior, social networking, and playing style in online and face to face bridge communities. In: NETGAMES, IEEE (2010) 1–2
27. Iosup, A., Lascateu, A.: Clouds and continuous analytics enabling social networks for massively multiplayer online games. In Bessis, N., Xhafa, F., eds.: Next Generation Data Technologies for Collective Computational Intelligence. Volume 352 of Studies in Computational Intelligence. Springer (2011) 303–328
28. Kim, K., Jeon, K., Han, H., Kim, S.g., Jung, H., Yeom, H.Y.: Mrbench: A benchmark for mapreduce framework. In: Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems. ICPADS '08, Washington, DC, USA, IEEE Computer Society (2008) 11–18
29. Chen, Y., Ganapathi, A., Griffith, R., Katz, R.H.: The case for evaluating mapreduce performance using workload suites. In: MASCOTS, IEEE (2011) 390–399
30. Pavlo, A., Paulson, E., Rasin, A., Abadi, D.J., DeWitt, D.J., Madden, S., Stonebraker, M.: A comparison of approaches to large-scale data analysis. In Çetintemel, U., Zdonik, S.B., Kossmann, D., Tatbul, N., eds.: SIGMOD Conference, ACM (2009) 165–178
31. Huppler, K.: Price and the tpc. In Nambiar, R.O., Poess, M., eds.: TPCTC. Volume 6417 of Lecture Notes in Computer Science., Springer (2010) 73–84
32. Kossmann, D., Kraska, T., Loesing, S.: An evaluation of alternative architectures for transaction processing in the cloud. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. SIGMOD '10, New York, NY, USA, ACM (2010) 579–590
33. Islam, S., Lee, K., Fekete, A., Liu, A.: How a consumer can measure elasticity for cloud platforms. [39] 85–96
34. Rabl, T., Poess, M.: Parallel data generation for performance analysis of large, complex rdbms. In Graefe, G., Salem, K., eds.: DBTest, ACM (2011) 5
35. Frank, M., Poess, M., Rabl, T.: Efficient update data generation for dbms benchmarks. [39] 169–180
36. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM **53**(4) (2010) 50–58
37. Villegas, D., Antoniou, A., Sadjadi, S.M., Iosup, A.: An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds. In: CCGRID. (2012)
38. Iosup, A., Yigitbasi, N., Epema, D.H.J.: On the performance variability of production cloud services. In: CCGRID, IEEE (2011) 104–113
39. Kaeli, D.R., Rolia, J., John, L.K., Krishnamurthy, D., eds.: Third Joint WOSP/SIPEW International Conference on Performance Engineering, ICPE'12, Boston, MA, USA - April 22 - 25, 2012. In Kaeli, D.R., Rolia, J., John, L.K., Krishnamurthy, D., eds.: ICPE'12 ICPE, ACM (2012)