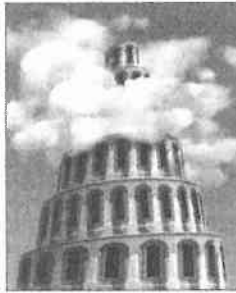


© IEEE 2017. The original document can be found using the DOI 10.1109/MIC.2017.1 or at

<http://ieeexplore.ieee.org/abstract/document/7839857/>



# Quality of Cloud Services: Expect the Unexpected

David Bermbach • TU Berlin

Here, the author presents a number of experiences from several years of benchmarking cloud services. He discusses how the respectively observed quality behavior would have affected cloud applications or how cloud consumers could use the behavior to their advantage.

In the last few years, cloud computing has found widespread adoption in companies of all sizes. A core focus of these cloud consumers is typically on cost savings, convenience of managed services, and on-demand capacity changes. The quality of cloud services, however, is usually taken “as-is”: based on documentation, advertisements, but also past experiences from a non-cloud world, cloud consumers typically have implicit assumptions. For instance, an eventually consistent storage system that claims to have triple replication in close-by datacenters with high-speed network interconnection can be assumed to show low millisecond staleness – that is, relatively good consistency behavior. As another example, virtual machines (VMs) that come in sizes S, M, L, and XL can be expected to grow in capacity for all resources – network bandwidth, disk storage volume, RAM size, CPU clock speed, or cores – when choosing a bigger instance type. However, in both examples this isn’t always the case – in fact, cloud consumers should always expect the unexpected.

This isn’t meant to imply that the unexpected is always bad for the cloud consumer. Actually, observable quality behavior is typically much better than what’s guaranteed, for example, when cloud consumers only plan for what’s guaranteed, they’ll never tap the full potential of cloud services. Furthermore, violations of guarantees might still occur. Therefore, not knowing about the quality of the cloud services used will generally either lead to unexpected negative surprises through (seemingly) obscure application behavior or to inefficiencies when cloud consumers design their applications only based on explicitly provided quality guarantees. The only way to avoid this

and to get insights into the actual quality of cloud services is through cloud service benchmarking (see the related sidebar).

Thus, here I report on a number of experiences from several years of benchmarking cloud services and briefly discuss how the respectively observed quality behavior would have affected cloud applications or how cloud consumers could use the behavior to their advantage. This article should be seen as a call: Don’t make assumptions, make experiments. For this purpose, I also sketch out how cloud consumers can use cloud service benchmarking in their application lifecycle.

### Performance of Virtual Machines

In 2011, Alexander Lenk and colleagues<sup>1</sup> ran a number of performance experiments on top of Amazon EC2 instances using the Phoronix test suite. Soon, they discovered that they had a twin peak distribution of compute performance results: For every benchmark in the suite, there were some machines that showed a very good performance while others showed a rather poor performance. Through in-depth analysis of results, they realized that the performance variance stayed constant over time and could also not be attributed to different instance sizes; instead, Amazon had obviously deployed two different CPU types (AMD Opteron and Intel Xeon). Depending on the CPU type, the machines either excelled at floating point or at integer operations. However, both types came with the same price tag. Furthermore, the performance difference could only be identified after having provisioned the instance. Obviously, this isn’t the anticipated behavior that a cloud consumer would expect from a cloud provider that offers a standardized product.



### Cloud Service Benchmarking

To fully understand cloud service benchmarking, first we must consider what a cloud service is, and how to determine its qualities.

#### What's a Cloud Service?

Much has been written about cloud computing, often focusing on delivery models or a cloud computing stack. However, with the availability of container technologies or lambda services, a differentiation into infrastructure as a service (IaaS) and platform as a service (PaaS) seems somewhat outdated. At the same time, Web APIs are widely used and NoSQL systems are much more similar to cloud storage services than the latter group is to virtual machines. For our purposes, a cloud service is, thus, a software system running in the cloud whose functionality is consumed programmatically by applications over Internet protocols. To applications, such cloud services appear like a black box, independent of the deployment model used, which is expected to adapt to application workloads while maintaining quality goals. Specifically, we consider an open source system such as Apache Kafka or Apache Cassandra, deployed on top of a compute service to be a cloud service as long as it's used/consumed like a service. This means that our understanding of cloud services is less

driven by the deployment model and more by the usage model.

#### What's a Cloud Service Quality?

A cloud service – that is, the software system behind the service interface, will confront the cloud consumer with a particular quality behavior: the cloud service might become unavailable, it might be slow to respond, or it might be limited with regards to the number of requests that it can handle. These are all examples of qualities – namely, availability, latency, or scalability – and an application using the respective service needs to have mechanisms in place to deal with these qualities (or, rather, deal with poor quality).

#### What's Cloud Service Benchmarking?

Cloud service benchmarking is a way to systematically study the quality of cloud services based on experiments. For this purpose, the benchmark tool creates an artificial load on the cloud service under test while carefully tracking detailed quality metrics. A key design goal of cloud service benchmarking is to mimic an application as closely as possible to get meaningful results; however, benchmark runs also aim to extensively stress the service, for example, through system load or even injected failures.

These benchmark results are an excellent example where cloud consumers could benefit from their knowledge on cloud service quality: depending on the performance requirements of the respective cloud application, the cloud consumer could simply start a new instance, run a short test, and then determine whether they wanted to use that instance for their application or whether to repeat the provisioning process.

Such unexpected behavior isn't a thing of the past. In 2015, we mentored a student project in which a group of master's students at Technische Universität Berlin ran a number of performance benchmarks on VMs. In their experiments, they compared an open stack-based SME cloud provider to Amazon's EC2. For their measurements, they used a subset of the Phoronix test suite to quantify CPU compute capacity, RAM, and disk throughput, but also network bandwidth between different VMs of all

sizes. What they discovered for the SME provider was that – across different VM sizes – compute power, memory, and disk throughput increased as expected – that is, an *M* instance generally showed better performance than an *S* instance. However, independent of the actual VM size, the available network bandwidth stayed constant. Considering the cost of different VM sizes, this leads to an interesting situation where for a number of applications it will be much more attractive to scale-out using the smallest VM type instead of scaling up; especially so for network-bound applications. These effects would never have been discovered without benchmarking.

#### Consistency of Cloud Storage Services

Cloud storage systems and services are typically replicated; many of them guarantee so-called eventual consistency. In such systems, an update operation terminates before writing all

replicas. This implies two things: first, that other clients can read outdated data while updates are being propagated; and second, that several clients might write the same data item concurrently, thereby leading to conflicts. Especially in the presence of failures (such as message loss or crashed instances), this inconsistency window – also called *staleness* – that is, the time during which outdated data might still be read, could become rather long. In general, applications can often tolerate staleness quite well; however, this becomes much easier if staleness is bounded. To our knowledge, even today there's no cloud provider that guarantees upper bounds for staleness. Therefore, in 2011, we developed a benchmarking approach for consistency and repeatedly measured consistency behavior of the Amazon Simple Storage Service (S3) over the years.<sup>2</sup> Basically, this approach aims to provoke the worst possible consistency behavior so as to obtain probabilistic



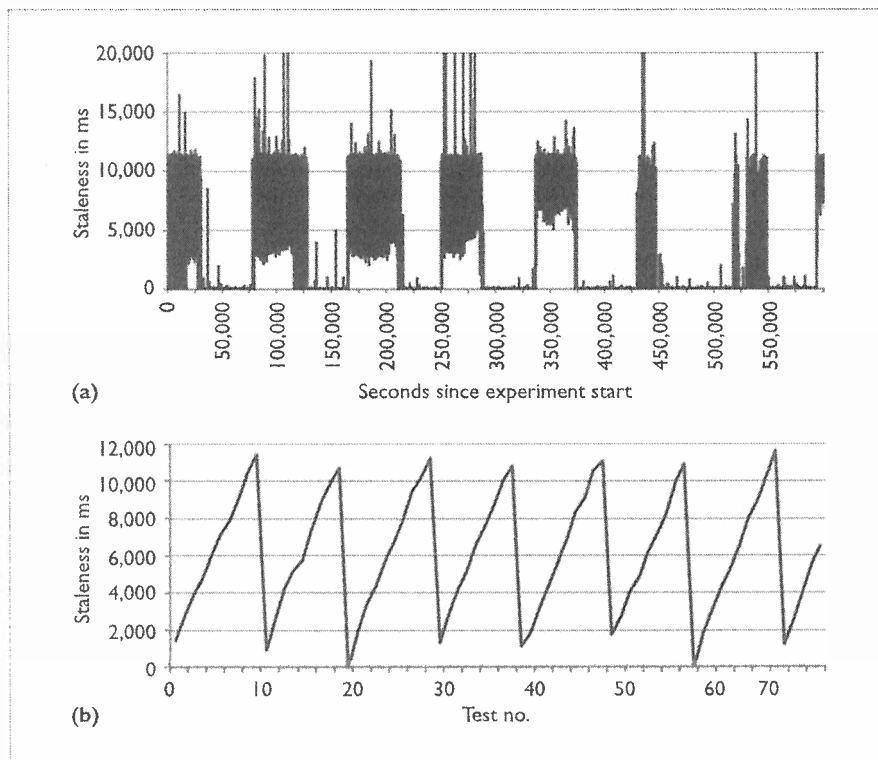


Figure 1. Consistency of Amazon Simple Storage Service (S3) in a one-week benchmark run in 2011. (a) At night, S3 showed much lower and more predictable staleness than during the day. (b) During the day, staleness of S3 followed a saw pattern. This pattern was independent of the interval between individual tests.

upper bounds on staleness, for example. The measurement approach comprises a number of distributed machines (for instance, 12 is a good number for three replicas) that continuously poll a target key. Another machine periodically updates that target key with the current timestamp and a version number (one test in Figure 1b). Correlation of values read and the respective current timestamp can then be used to determine the staleness. Furthermore, this data also can be used to determine the probability of reading stale data as a function of the duration since the last update.

S3 guarantees eventual consistency based on at least three replicas located in adjacent datacenters. What could be expected, hence, were staleness values in the lower two-digit millisecond range. However, in our first (repeated) experiments in 2011, we found that while S3 had acceptable staleness at

night, it followed an obscure saw pattern during the day. Figures 1a and 1b, taken from previous work,<sup>2</sup> show this behavior measured during a one-week benchmark run: During the day, the first update has a 2-second staleness, the second one a 4-second staleness, and so on until it drops back down after close to 2 minutes and starts all over again. Of course, we contacted Amazon about this behavior and also continued to benchmark S3 consistency behavior over the years: Not only was the initial behavior totally unexpected – until our last benchmark run in late 2013, it continued to change significantly (see Figure 2<sup>3</sup>), thus providing further proof for our “expect the unexpected” mantra. Without going into further details, suffice it to say that dealing with inconsistencies at the application level isn’t too difficult – unless there’s no information on the quality behavior of underlying cloud services.<sup>3</sup>

## Security of Cloud Storage Services

Especially when dealing with sensitive data in cloud environments, security becomes a key design goal – particularly for data-in-transit security, where data are encrypted and hashed before being sent over the Internet. This, however, can be expected to come with a performance impact – which has largely been neglected by researchers so far: either researchers focus on security so that performance impacts are largely disregarded, or they focus on performance, then ignoring security or choosing the weakest option available.

In recent experiments, I’ve worked with colleagues to benchmark how enabling data-in-transit security (for example, based on TLS) affects the performance of cloud storage services. Interestingly, though, there’s no clear result, as the impact completely depends on the concrete system. For instance, in previous work,<sup>4</sup> we described how Apache Cassandra configurations with TLS might, in fact, outperform unsecured configurations (essentially, this means that the natural performance variability of cloud resources exceeds and hides the performance impact of TLS); this, however, depends on the respective configuration and setup details. Amazon’s DynamoDB service, on the other hand, shows no performance impact at all – aside from computation overheads on the application machines, the performance overhead is shouldered and paid for by Amazon. On the other hand, we’ve seen in recent experiments with Apache HBase that enabling data-in-transit security could have a catastrophic impact on performance, thereby also severely limiting scalability.<sup>5</sup> For example, we could observe that a 12-node HBase cluster with data-in-transit security enabled can sustain approximately the same throughput as an unsecured 6-node cluster.

For application developers, this should have a strong effect on the



service-selection process. For instance, HBase should be avoided if security is necessary in cloud deployments. On the other hand, a hosted service might be an excellent choice where maximum security essentially comes for free as long as you trust the cloud provider.

### Availability of Web APIs

As a completely different example of cloud services, we recently ran a three-month experiment where we benchmarked performance and availability of Web APIs which, as I previously described, we also consider cloud services due to their similarity from a service consumption perspective. A key aspect of our experiment<sup>6</sup> was the geodistribution of clients: because Web and mobile applications are inherently distributed – either through a global user base or through the geomobility of individual users – we deployed our benchmarking clients all over the world. For the experiment, we selected 15 hand-picked Web APIs so as to cover a wide variety of application areas, countries, provider sizes, and so on. Each of the benchmarking clients periodically called all 15 Web APIs over both HTTP and HTTPS, and also pinged the API host. For these calls, we collected detailed results and thus could track latency and availability.

What we expected to find in the results was a performance variance depending on the geolocation of the client – this was typically the case. However, what we also expected was that availability would be comparable across locations. This was absolutely not the case. We were surprised to find that there were several APIs that had an availability of less than 50 percent for most of the days of our experiment – however, this was true only in some regions while they were fully available in others. For an unknown reason, some APIs don't have the same availability across geographic regions so that end users of mobile applications built on top might be confronted with negative

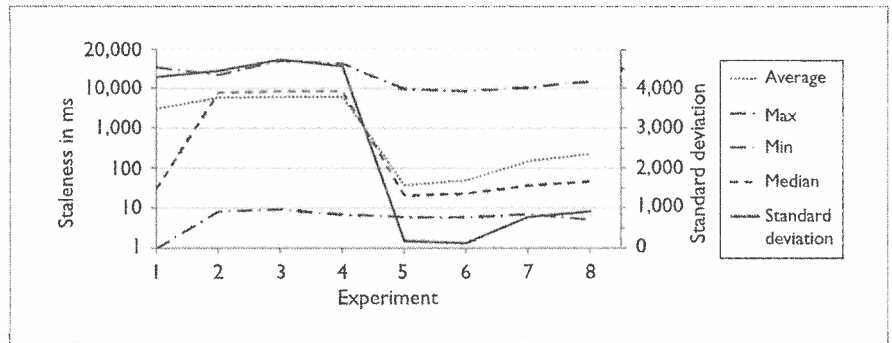


Figure 2. Consistency behavior of Amazon S3, as determined through one-week benchmark runs in 2011–2013. Throughout the benchmark runs, behavior changed significantly.

surprises. Another curious behavior was that there was approximately a 70 percent chance of the HTTPS endpoint of an API being available while the HTTP endpoint of the same API wasn't (and vice versa). This indicates that Web API providers often have separate front-end servers per protocol and only share the backend services.

Both results can be dangerous for application developers if they aren't known. However, they also can be leveraged, for example, by trying the respective other protocol in case of unavailability or by tunneling requests through additional backend servers in other geographic regions.

### Cloud Service Benchmarking for Developers

Now that we've seen how cloud services show unexpected behavior again and again, when and how should application developers use cloud service benchmarking?

Generally, a cloud migration or the development of a cloud-native application will begin with an initial assessment phase, where the developers decide on the target runtime environment but also on the set of cloud services that their application will use. In this phase, it's useful to select existing benchmark implementations that are as similar as possible to the application workload. Developers should then use these benchmark tools to better under-

stand the quality of all options. Of course, there are often interdependencies – for instance, developers might need cloud services that are offered by only a single provider, or a federated setup<sup>7</sup> might be desirable. This should lead to an environment of handpicked cloud services, along with initial ideas for dealing with quality problems.

Afterward, in the (initial) development phase we would recommend implementing micro-benchmarks as well for the application itself – similar to unit tests, benchmarks for testing non-functional properties should be part of the build process. This approach is especially well-suited for microservice-based applications where modules can be benchmarked individually. During this phase, it also makes sense to periodically reassess the quality of underlying cloud services. Finally, when the application goes into production, underlying cloud services should be carefully monitored using both monitoring, periodic benchmarking, or indirect monitoring,<sup>2</sup> where business key performance indicators (KPIs) gauge for quality changes in the underlying cloud services. Whenever something unusual happens, developers should reassess the quality of the cloud services used, but also adapt their deployment decisions by, for example, switching providers. Of course, actually implementing this approach in practice comes with a number of challenges;



however, these are beyond the scope of this article.

In all these examples, we have seen how completely unexpected behavior recurs in all kinds of cloud services. Application developers should, therefore, never assume that cloud services behave like traditional on-premises environments – instead, developers should expect the unexpected and prepare for it. This, however, is only possible through cloud service benchmarking: Don't make assumptions, make experiments.

There are a number of open challenges that would benefit from future research efforts. The first is that benchmarks are typically designed for reuse. However, especially in the context of custom microservices, currently it's unclear how benchmarks that are part of the build process can be generalized and reused. After all, a specific microservice is rather unique in its nature so that developing a "standard" benchmark that doesn't only test a minimum subset of features is quite challenging. The second aspect is moving from fine-granular benchmarks (or even micro-benchmarks) to more high-level benchmarks. After all, application developers are often more interested in the overall quality of an entire cloud platform than in assessing individual cloud services – or even worse: of small subsets of a service such as disk throughput of a VM. In this area, identifying suitable, realistic application workloads but also again the portability of benchmarking tools is an unsolved major challenge. The third challenge is on developing benchmarks that assess multiple qualities at the same time – currently, most benchmarks measure only one quality, usually performance. □

### Acknowledgments

I thank Steffen Müller, Frank Pallas, Stefan Tai, and Erik Wittern for the joint work leading to the



experimental results used as a basis for this article.

### References

1. A. Lenk et al., "What Are You Paying for? Performance Benchmarking for Infrastructure-as-a-Service Offerings," *Proc. IEEE Int'l Conf. Cloud Computing*, 2011, pp. 484–491.
2. D. Bermbach and S. Tai, "Benchmarking Eventual Consistency: Lessons Learned from Long-Term Experimental Studies," *Proc. IEEE Int'l Conf. Cloud Eng.*, 2014, pp. 47–56.
3. D. Bermbach, "Benchmarking Eventually Consistent Distributed Storage Systems," PhD thesis, Dept. of Economics and Management, Karlsruhe Inst. of Technology, 2014.
4. S. Müller et al., "Benchmarking the Performance Impact of Transport Layer Security in Cloud Database Systems," *Proc. IEEE Int'l Conf. Cloud Eng.*, 2014, pp. 27–36.
5. F. Pallas, J. Günther, and D. Bermbach, "Pick Your Choice in HBase: Security or Performance," *Proc. IEEE Int'l Conf. Big Data*, to appear.
6. D. Bermbach and E. Wittern, "Benchmarking Web API Quality," *Proc. Int'l Conf. Web Eng.*, 2016, pp. 188–206.
7. T. Kurze et al., "Cloud Federation," *Proc. Int'l Conf. Clouds, Grids, and Virtualization*, 2011, pp. 32–38.

David Bermbach is a senior researcher in the Information Systems Engineering research group of TU Berlin. His research interests include cloud service benchmarking, cloud applications, and IoT platforms, but also middleware and distributed systems in general. Bermbach has a PhD with distinction in computer science from Karlsruhe Institute of Technology. Contact him at [db@ise.tu-berlin.de](mailto:db@ise.tu-berlin.de).


Read your subscriptions through the myCS publications portal at <http://mycs.computer.org>.



## 2017 B. Ramakrishna Rau Award Call for Nominations

*Honoring contributions to the computer microarchitecture field*

**New Deadline: 1 May 2017**



Established in memory of Dr. B. (Bob) Ramakrishna Rau, the award recognizes his distinguished career in promoting and expanding the use of innovative computer microarchitecture techniques, including his innovation in compiler technology, his leadership in academic and industrial computer architecture, and his extremely high personal and ethical standards.

**WHO IS ELIGIBLE?** The candidate will have made an outstanding innovative contribution or contributions to microarchitecture, use of novel microarchitectural techniques or compiler/architecture interfacing. It is hoped, but not required, that the winner will have also contributed to the computer microarchitecture community through teaching, mentoring, or community service.

**AWARD:** Certificate and a \$2,000 honorarium.

**PRESENTATION:** Annually presented at the ACM/IEEE International Symposium on Microarchitecture

**NOMINATION SUBMISSION:** This award requires 3 endorsements. Nominations are being accepted electronically: [www.computer.org/web/awards/rau](http://www.computer.org/web/awards/rau)

**CONTACT US:** Send any award-related questions to [awards@computer.org](mailto:awards@computer.org)

**[www.computer.org/awards](http://www.computer.org/awards)**