How is the Weather tomorrow? Towards a Benchmark for the Cloud

Carsten Binnig Donald Kossmann Tim Kraska Simon Loesing

Systems Group, Department of Computer Science, ETH Zurich
{firstname.lastname}@inf.ethz.ch

ABSTRACT

Traditionally, the goal of benchmarking a software system is to evaluate its performance under a particular workload for a fixed configuration. The most prominent examples for evaluating transactional database systems as well as other components on top (such as a application-servers or web-servers) are the various TPC benchmarks.

In this paper we argue that traditional benchmarks (like the TPC benchmarks) are not sufficient for analyzing the novel cloud services. Moreover, we present some initial ideas how such a new benchmark should look like that fits better to the characteristics of cloud computing (e.g., scalability, pay-per-use and fault-tolerance). The main challenge of such a new benchmark is to make the reported results comparable because different providers offer different services with different capabilities and guarantees.

1. INTRODUCTION

Traditionally, the goal of benchmarking a software system (called *system under test or short SUT*) is to evaluate its average performance under a particular workload. The most prominent examples for evaluating transactional database systems as well as other components on top (such as a application-servers or web-servers) are the various TPC-benchmarks that define workloads derived from different real-world application scenarios (e.g., TPC-H for OLAP [14], TPC-C for OLTP [15], or TPC-W [13] for an e-commerce application).

All the TPC-benchmarks require that the system under test is deployed in a managed environment using a fixed configuration (e.g., a static set of software- and hardware-components) which is described in a full disclosure report. Consequently, the primary metrics of all the TPC-benchmarks reflect the average performance of a static non changing system. Another primary metric that is often reported by the TPC-benchmarks are the total costs of ownership for such a static system over its lifetime (i.e., the costs for hard- and software, maintenance as well as administration). Moreover, as all TPC-benchmarks focus on transactional database systems they force these systems to provide the ACID properties as well as different levels of isolation as defined by the SQL standard.

Recently, the cloud computing paradigm [3] has lead to novel solutions for storing and processing data in the cloud. Examples are Amazon's S3, Google's Bigtable, or Yahoo's PNUTS. Compared to traditional transactional database systems, the major advantage of these novel storage services is their elasticity in the face of changing conditions. For example, in order to adapt dynamically to the load, cloud providers automatically allocate and deallocate resources (i.e., computing nodes) on the fly while offering a pay-as-you-go computing model for billing. However, in terms of query processing the functionality provided by these novel storage services is often far away from a full SQL support.

Another important difference of the cloud storage services compared to the traditional transactional database systems are the consistency guarantees provided by these services. In order to offer fault-tolerance and high-availability, most providers replicate the data within one data or even across data centers. Following the CAP theorem [10], it is not possible to provide availability and strong consistency (as defined by the ACID properties) together in the presence of network failures. Consequently, most cloud providers sacrifice strong consistency for availability and offer only some weaker forms of consistency (e.g., Amazons's S3 guarantees only eventual consistency).

In addition to the storage services, many cloud providers also offer additional services that allow developers to bring the complete application stack into the cloud while offering the same advantages as for the storage services (pay-per-use, scalability, fault-tolerance). Examples for these services are Amazon's EC2 or Google's App Engine which provide an application server in the cloud or Amazon's SQS which provides a messaging service.

In this paper we want to start a discussion why traditional benchmarks (like the various TPC-benchmarks) are not sufficient for analyzing these novel cloud services. Moreover, we present some initial ideas how a new benchmark should look like that fits better to the characteristics of the cloud (scalability, pay-per-use, fault-tolerance). The main challenge of such a new benchmark is to make the reported results (i.e., the primary metrics) comparable because different providers offer different services with different capabilities and guarantees of these services (e.g., the consistency guarantees of the storage services).

An important difference to most existing benchmarks is that a new cloud benchmark should not require to use a static configuration of software- and hardware-components because dynamic allocation and deallocation of resources as well as the pay-as-you-go model are the inherent features of these services. Consequently, we think that a new benchmark for the cloud must report different metrics than the existing benchmarks: Instead of measuring the average performance of a static system under maximal load, the new metrics should reflect the ability of the cloud services to adapt to a

DBTest'09, June 29, 2009, Providence, Rhode Island, USA.
Copyright 2009 ACM 978-1-60558-706-6/09/06 ...$5.00.

changing load with regard to performance and costs. Moreover, an additional metric should also cover the robustness of these services against failures of single nodes as well as the outage of complete data centers.

We also think that a new cloud benchmark should focus on analyzing the complete stack of web applications instead of introducing micro benchmarks for single cloud services. Consequently, a new cloud benchmark should not only analyze the storage services in the cloud but also the other cloud services (e.g., application servers, message systems). However, compared to the existing TPC-W benchmark, a new benchmark for the cloud should define additional Web 2.0 like interactions and be based on modern technologies such as AJAX which change the access patterns of a web application.

To the best of our knowledge, currently there exists no benchmark for cloud services with similar goals as those that we discussed before. However, there exists a study [9] which evaluates the different cloud services of Amazon in terms of cost and performance but it does not provide a general benchmark which allows the comparison of different cloud providers. Another work [7] compares the cloud computing technology with current transactional database systems and proposes a list of comparison elements. However, they also do not discuss how a benchmark for comparing the cloud services of different providers should look like.

Thus, the contributions of this paper are:

- We first reiterate the most important characteristics of cloud services and derive a list of requirements that a new benchmark for these cloud services should fulfill (Section 2).
- Afterwards, we analyze the existing TPC-W benchmark and its metrics and discuss why the TPC-W benchmark does satisfy the requirements for benchmarking cloud services (Section 3)[1].
- As a last contribution, we discuss some initial ideas for such a new cloud benchmark that tackles the shortcomings of the TPC-W benchmark (Section 4).

2. CLOUD COMPUTING

Cloud computing allows users to access technology-enabled services from the Internet ("in the cloud") without owning the technology infrastructure that supports them [11]. The range of services today varies from basic infrastructure services, such as providing storage space, to rather specialized services for payment, identity authentication and others. This section reiterates the advantages of using cloud services. It provides an overview of the services offered today and sketches the requirements for a benchmark for the cloud.

2.1 Why Cloud Computing?

Generally, three different types of cloud services are differentiated. At the core, *Infrastructure as a Service (IaaS)* provision resources such as servers (often in form of virtual machines), network bandwidth, storage, and related tools necessary to build an application environment from scratch (see Figure 1). The business model is *pay-per-use*. Thus the users do not need to make an investment upfront and pay for the hardware resources they consume. Furthermore, most providers guarantee virtually infinite resources. Hence, resources are provided on demand and there is no need to plan far ahead for provisioning. Although infrastructure services provide the highest flexibility, developers still have to deal with low level details such as maintaining virtual machines or load-balancing.

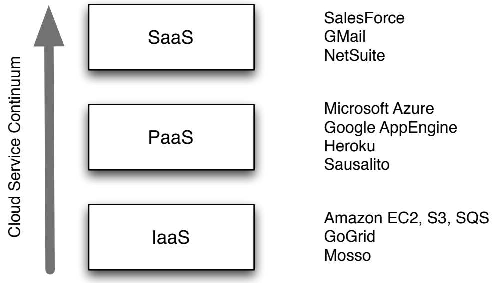

Figure 1: Cloud Service Continuum

Platform as a Service (PaaS) often build on top of IaaS and provide a higher-level environment, a platform, on which developers write customized applications. Those platforms are managed by the service provider. The maintenance, load-balancing and scale-out of the platform are done by the service provider and the developer can concentrate on the main functionalities of his application. In exchange for the built-in scalability the developer is accepting some restrictions on the type of software he can write. Again the business model is *pay-per-use* and the platform is virtually infinitely scalable.

Finally, *Software as a Service (SaaS)* refers to special-purpose software made available through the Internet. Well-known examples of SaaS are GMail or Salesforce. However, those services are not suited for building individual applications and are excluded in the further discussion.

The success of cloud computing for all service classes is based on economy of scale. The cloud provider can offer services to millions of users at a lower price than users accommodating these services themselves. In addition to the price, the quality of the service is a major incentive for using cloud services. In particular, the cloud provider is responsible for guaranteeing high availability and reliability. Users expect almost 100% availability and (virtually) constant response times independent of the number of concurrent users. Furthermore, users of cloud services do not need to worry about scalability because the offer is virtually infinite. The IT cost grows linearly with the business rather than step-wise as is the case for traditional computing in which businesses need to buy hardware in the granularity of machines (rather than CPU cycles). In summary, the goal of cloud computing is to provide more for less.

2.2 Cloud Services Today

By today, a huge variety of services exists offering different infrastructure or platform services.

IaaS. The most flexible form of IaaS are server-hosting services like Amazon's Elastic Computing Cloud (*EC2*) or GoGRID. These services allow to rent virtual machines (*VM*) running the different flavors of Unix/Linux or Windows. Those virtual machines can be freely configured and enable to run almost every kind of software.

Although the general concept of renting a virtual machine is similar for all providers, the services differ in the way they are priced, how they persist data over failures, SLA, geographical location, and the tooling they provide. Most critical for developing an application is how persistent data is stored and handled. Generally the virtual machines loose their data if they get shut down (e.g. by the user or by a hardware failure). In the case of GoGRID, the developer himself has to take care of persistence. To help the developer,

[1] We did not compare to the TPC-App benchmark because this benchmark focuses more on Web-Services and not on the complete web-application stack.

GoGRID offers a persistent (single data center replicated) network attached storage for backups. Amazon on the other hand offers two additional services to store data persistently: Amazon's *Simple Storage Service (S3)* and the *Elastic Block Store (EBS)*. S3 can be used by several clients simultaneously, is highly reliable due to distributing the data across data centers, provides eventual consistency guarantees and is optimized for reads of larger files. On the other hand, the Elastic Block Store is a special storage service which can only be accessed by a single virtual machine at a time, is optimized for writes, but only replicated in one data center. Using one or the other storage service results in completely different guarantees and limitations. With S3, the data is always readable - even in the case of network partitioning or data center failure, there is no restriction in the scale-out. However, the data itself is just eventual consistent. With EBS the data is highly consistent but at the same time can only be accessed by one VM. Thus, the scalability is limited.

Additionally, a huge range of open source and commercial solutions exists to build own infrastructure services. For example, Cassandra [8] or HBase [2] are solutions which could be deployed on top of EC2 or self-owned machines to provide persistent storage. Again, the different solutions vary in the form of consistency guarantees they provide, the ability to replicate across data centers, performance, index support etc.

A variety of additional IaaS offerings such as queue services or content delivery exists which can help to build distributed, highly available applications. To list them all is out of scope of this paper.

PaaS. For PaaS, the most prominent examples are Google's App Engine and Microsoft's Azure platform. Google's App Engine hosts python programs in a highly scalable manner. Similar to IaaS, data persistence is one of the critical differentiators between the different platforms. Google provides a datastore API allowing to permanently persist data. The system behind the datastore is most likely MegaStore [6]. With this API, the developer can take advantage of the notion of transactions and even a simple query language. However, data is required to be grouped into so-called entity groups. Transactions are only possible inside such entity group. Although there is no restrictions on the size of an entity group, the scalability is limited as per entity group only one transaction can operate at a time. That is, all transactions are serialized.

The Azure Service platform is Microsoft's PaaS offering. Instead of Python, Azure is based on the .Net language. Similar to Google's App Engine it has a dedicated API to store and retrieve data called SQL Services. The underlying system for these SQL Services is Microsoft SQL Server. Although not all functionalities of Microsoft SQL Server are exposed via the API, the user can run transactions and use a restricted SQL query language. Similar to Google's App Engine, the user has to partition the data manually into so-called containers. Transactions and queries are restricted to one container at a time. In contrast to Google's App Engine, inside one container several transactions can run simultaneously. But containers are restricted with regard to their size (i.e., 2GB).

There also exists a wide variety of startups offering different platforms for different languages like XQuery or Ruby on RailsP [1, 12]. Again, next to the language and offered libraries, all the providers mainly vary in how they handle persistent data.

2.3 Requirements to a Cloud Benchmark

As indicated above, today's cloud services differ among others by cost, performance, consistency guarantees, load-balancing, caching, fault tolerance, SLA and programming language. System architects and developers are confronted with this variety of services and trade-offs. Hence, the purpose of a cloud benchmark should be to help the developer when choosing the right architecture and services for their applications.

Features and Metrics. Arguable, the main advantages of cloud computing are scalability, pay-per-use and fault-tolerance [11, 3]. Despite the promises of cloud providers those features are often differently fulfilled. For example, most cloud providers claim to provide nearly infinite scalability for their services, but it is not self-evident that the combination or some of the limitations of one or more services does not yield a scalability limit (e.g., the limitations on the persistent storage).

Furthermore, price plans and the granularity of the pricing lead to different overall costs. For example, for PaaS pricing is typically based on CPU utilization. A web application which has to support a single transaction per hour, is priced exactly for this single transaction. In contrast, for IaaS virtual machine instances are often priced on a per hour basis. The costs for a VM instance, however, are the same regardless if one or 1000 transactions are performed per hour. In economics, this is often referred to as lot-size problem. The interested reader might notice, that it is possible to see the classical architecture with self-owned hardware as an extreme case having really high lot-sizes.

Finally, fault-tolerance differs significantly between providers. The number of faulty services the system can resist without user notice, or single data-center vs. multi data-center replication are just some examples.

As mentioned before, the traditional benchmarks are mainly concerned with performance and cost of static systems. Those metrics still have relevance for the cloud applications but we need different ways for measuring them for scalable (i.e., dynamic) systems where resources come and go. Moreover, a benchmark for the cloud should additionally test the cloud-specific features (scalability, pay-per-use and fault-tolerance) and provide appropriate metrics for them.

Architectures. By today, there exists no agreement on the right architecture and use of cloud services. Different services can be used and extended in different ways to achieve the same goal. For example, it is not obvious if using S3 with indexes built on top is superior to using SimpleDB. It is also not obvious, if S3 is better than a similar offering hosted in the EC2 environment or if a platform as a service is the better overall solution.

Furthermore, it is often the case that services from different providers can not be freely combined (e.g., Microsoft SQL Data Services with Google's App Engine) and they have completely different guarantees. It is more important to know how different services play together rather than to know which provider is particularly good in just a single aspect (e.g., a key-value store).

As imposing one architecture variant is not reasonable, a cloud benchmark should be general enough to cover the different architectural variants. Furthermore, the complete application stack should be measured instead of micro-benchmarking single services.

Nevertheless, different service architectures are not always directly comparable. For example, one of the most critical differentiators between the cloud providers is caused by the CAP theorem [5]. The CAP theorem states that it is not possible to achieve consistency, availability and tolerance against network partitioning at the same time. In addition, it can be seen that achieving strong consistency in a highly distributed system is more expensive (in terms of latency and throughput) than weak consistency and hence often restricts the scalability. As it is not possible to have it all, cloud solutions position themselves somewhere in the design space of relaxed consistency, high scalability, and high availability. A benchmark should on the one hand consider these different design spaces to avoid comparing apples and oranges, while on the other hand it should not to force all solutions into a single setup.

3. TPC-W AND ITS PROBLEMS

The TPC-W benchmark specifies an online bookstore that consists of 14 web interactions allowing to browse, search, display, update and order the products of the store. The system under test consists of an application server implementing the business logic responsible for answering every HTTP request and a persistent storage usually realized using a transactional database system.

In order to generate the workload, the TPC-W specifies a remote browser emulation (RBE) system which automatically simulates an arbitrary number of users sending requests for single web interactions to the system under test. The goal of the RBE is a realistic simulation of the browsing behavior of different users. To issue different workloads against the system under test, the TPC-W benchmark defines three different mixes: browsing, shopping and order mix. A particular mix determines for every user session a sequence of web interactions based on a varying ratio of browse and order operations. Browse operations only read data whereas order operations execute data updates. Benchmarking with the different mixes shows which impact a varying number of update operations has on the performance of the system.

The TPC-W benchmark is designed to test the complete application stack and does not make any assumptions on the technologies and software systems used. Thus, two essential requirements are already fulfilled by the TPC-W. However, trying to use the TPC-W as it is for benchmarking the cloud reveals some problems.

First, by requiring the ACID properties for data operations it becomes obvious that the TPC-W has been designed for transactional database systems. As already discussed earlier, cloud systems usually do not offer such strong consistency constraints because most web-based applications only require lower levels of consistency [16]. As a consequence existing TPC-W implementations for the cloud (e.g., [4]) are not conform to the specification and results of different implementations hard to compare.

Second, the primary metric used by the TPC-W is the number of web interactions per second (WIPS) that the system under test can handle. By scaling the number of emulated browsers, the number of requests and the load on the system can be increased. This is done as long as 90% of the web interaction response times does not exceed a specified amount of seconds. In such a situation the benchmark run is considered valid. The performance of a system is then reported as the highest number of WIPS reached in a valid benchmark run. Although WIPS is useful in the context of a static system it is not for adaptable and scalable systems. In an ideal cloud computing setting an increasing load would always be compensated by adding new processing units to the system and thus the number of WIPS would continuously increase. Consequently it is not possible to report the maximum WIPS value and the main metric is useless for the cloud.

Third, the second metric of the TPC-W is the ratio of costs and performance: $/WIPS. The pricing is based on the total cost of ownership of the system under test including software, hardware, maintenance and administration expenses (for 3 years). These overall costs are then divided by the maximum number of WIPS to calculate the $/WIPS. For a cloud benchmark two problems arise: First, as discussed earlier in the context of cloud computing no maximum number of WIPS exists. Thus, there exists no fixed load for which the overall cost can be calculated. Secondly, different price-plans and the lot-size problem prevent to calculate a single $/WIPS number. Instead, the $/WIPS may vary extremely depending on the particular load.

Fourth, the latest release of the TPC-W specification dates back to 2002. Considering the technical evolution of web applications in the last years, the TPC-W became outdated and does not reflect modern access-paths such as those generated by Web 2.0 like interactions (e.g., user generated content or AJAX).

Finally, the TPC-W benchmark lacks of adequate metrics for measuring the features of cloud systems like scalability, pay-per-use and fault-tolerance.

In the next section we present some ideas, including new metrics, for testing the performance of applications running in a cloud environment.

4. IDEAS FOR A NEW BENCHMARK

As discussed in the Section before, there are several reasons why the existing TPC-W benchmark is not sufficient for analyzing cloud services. In this Section, we present initial ideas towards a new cloud benchmark that better fits to the characteristics of these services: We first discuss the big picture of a new cloud benchmark and then present details on the different possible configurations. Finally, we introduce new metrics for analyzing the scalability, the costs, and the fault tolerance of cloud services.

4.1 Big Picture

We propose that a new cloud benchmark should be based on a e-commerce scenario (i.e., a web-shop) and define web interactions as benchmark drivers similar to the TPC-W benchmark. Thus, the benchmark should allow the evaluation of the complete application stack rather than having multiple micro-benchmarks for single services of different providers.

As discussed before, a fundamental difference of a cloud benchmark compared the TPC-W benchmark are the reported primary metrics. While the TPC-W benchmark analyzes the average number of web interactions per second of a static system under maximal load, a new cloud benchmark should analyze the ability of a dynamic system to adapt to a changing load (including peaks) in terms of scalability and costs. Moreover, another goal is to test to the assumption of infinite scalability of an application in the cloud. Consequently, different from the TPC-W benchmark which uses a ramp-up phase to find the maximal web interactions per second (WIPS) that can be issued against the system under test, a cloud benchmark explicitly needs to vary the WIPS during the benchmark execution. In addition to these metrics, the benchmark should also report metrics which represent the tolerance of the cloud services against failures. More details about these new metrics and the benchmark execution will be given in the following sections.

Another important issue when benchmarking cloud services is the locality of the emulated browsers (i.e., the test drivers) that trigger web interactions during benchmark execution. Cloud providers often replicate data over different data centers for availability but also performance reasons (due to locality). In order to get a fair comparison of the benchmark results, the emulated browsers should run in different locations (world wide). By doing this, we can achieve that the benchmark results are not biased due to the location where the test driver is running (i.e., we expect to get better results as closer the test driver gets to the data center due to Internet latency). A solution to this problem is to run the test drivers on a cloud infrastructure of a provider which supports location based installations (such as Amazon).

Moreover, compared to the TPC-W benchmark a new cloud benchmark should not require the data storage services need to provide the strong transactional ACID guarantees for all web interactions. Instead, we think that the consistency level is a configuration property of the benchmark which can be set to evaluate different application scenarios. Again, more details about the different consistency settings will be given in the next section.

Finally, independent of benchmarking cloud services, we suggest that in addition to the traditional web interactions defined by the TPC-W benchmark for browsing and ordering products of a web shop, a new benchmark should comprise web interactions that resemble the access patterns of Web 2.0 like applications. One example is to add web interactions that allow users to write and read reviews of individual products. Another idea, would be to add web interactions that allow user communities to exchange the latest news about certain products.

Moreover, on the same line of arguments, web 2.0 applications often include multi media content (audio files, video files, pictures) which can be accessed by users. This content produces heavy load on the servers which host that content. Thus, we suggest that individual web interactions of a new benchmark should refer to such content(e.g., by adding audio and video samples for the products of a web-shop).

4.2 Benchmark Configurations

Similar to the TPC-W benchmark, we suggest that individual runs of the benchmark can use different settings for the scale of the database as well as choose between different web interaction mixes (such as the Browsing mix, the Shopping mix, or the Ordering mix). However, as discussed before, in addition to these configurations, we believe that the consistency level is another orthogonal parameter that can be varied by the cloud service provider for the benchmark execution.

This consistency configuration parameter tackles the problem that cloud providers usually offer different consistency guarantees for their storage services ranging from the weak BASE guarantees (Basically Available, Soft-State, Eventually Consistent) to the strong transactional ACID (Atomicity, Consistency, Isolation, Durability) in order to support different kinds of web applications (e.g., ranging from non transactional applications to share personal data to more transactional enterprise resource planning applications). Consequently, in order to analyze the spectrum of different consistency guarantees using a cloud benchmark on the one hand and the need to produce comparable benchmarking results on the other hand, we propose that such a new benchmark can choose between three different levels of consistency:

- **Low:** All web interactions use only the BASE guarantees.
- **Medium:** The web interactions use a mix of consistency guarantees ranging from BASE to ACID. For example, user reviews can use the weak BASE guarantees while orders for products need the strong transactional ACID guarantees (or at least a higher SLA for the consistency).
- **High:** All web interactions use only the ACID guarantees.

Consequently, if a storage service of a particular cloud provider does not provide the desired consistency level, either the benchmark can not be executed using this consistency level or the benchmark implementation must add additional functionality to the application layer to provide the necessary consistency guarantees. We suggest also that a new benchmark provides a test suite to analyze if the system under test satisfies the consistency guarantees that are needed by the given benchmark configuration.

The metrics that are reported by the cloud benchmark (see next section) thus always refer to a certain setting including database scale, web interaction mix, and consistency level.

4.3 Metrics

As discussed earlier, reporting the maximum number of WIPS and the $ per WIPS is not useful in the context of cloud deployments. Instead, metrics are needed that explicitly focus on the characteristics of cloud computing: scalability, pay-per-use and fault-tolerance. In the following, we sketch out different metrics which particularly try to measure the dynamic aspects of the cloud.

Scalability. Ideally, cloud services should scale linearly and infinitely with a constant cost per WI. Service restrictions, consistency requirements, price plans and physical limitations can prevent the perfect scaling. Hence, one of the main figures of a cloud benchmark should report the scalability of the system.

We suggest to measure the scalability by increasing the issued web-interactions per seconds over time and continuously counting the web-interaction which are answered in a given response time interval (e.g., 1 second). That is, the number of web interactions issued against the system increases with a pre-set rate. Ideally, the system scales linearly and answers all issued WIPS in the allowed time-frame. In Figure 2, the blue-line shows the issued WIPS. This line also corresponds to the perfect scaling. If the system does not scale perfectly, more and more WI will not be answered in the giving response time (RT) and hence less *WIPS in RT* are processed (indicated as the red line in Figure 2). We suggest to measure this deviation by using the correlation coefficient R^2 or by determining the parameters of a power function.

The correlation coefficient R^2 is a value between 0 and 1, indicating how well a predicting function fits the data. A value of 1 stands for perfect prediction whereas a value of 0 indicates a constant behaviour. By setting the predicting function to the perfect case (the linear scaling) the value indicates how strong the measured scalability deviates from perfect scalability.

Alternatively, we suggest to use non-linear regression to determine the parameter b of a power function of the form $f(x) = x^b$. Again, b is a value between 0 and 1, where 1 indicates perfect linear scaling. This method has the advantage that the function can be used to make predictions about the scalability for a given workload. But at the same time, the method assumes a certain underlying function which might not properly reflect the real behaviour.

Which of these (or other) methods is the most appropriate one to measure the scalability is open for discussion.

Another difference to TPC-W is the running time of the benchmark. Assuming that perfect scaling does not exist, we suggest to define the end of the benchmark as the time when the difference between the *Issued WIPS* and the *WIPS in RT* exceeds a predefined limit. Still, getting to this value might take arbitrarily long. To overcome this issue, the benchmark should additionally define a minimum execution time for which the results are assumed to be sufficiently significant.

Cost. In the lines of the TPC-W benchmark, we propose to measure the cost in dollars per WIPS ($/WIPS). Thus, the cost of running the system (including all self-owned infrastructure and administration cost) is divided by the current WIPS rate. Ideally, the cost is constant independently of the current scaling factor of the system (perfect utility pricing). However, lot-sizes or price plans might cause variations of the $/WIPS in respect to the current scaling factor. Figure 3 demonstrates the effect of lot-sizes on the cost per WIPS. For example, the jump in the cost could be caused by additional EC2 instances which were required to handle the increased load. An EC2 instance comes at a fixed cost which might considerably increase the cost from one moment to another.

To cover this dynamic behaviour we suggest measuring not only the average cost per WIPS but also the standard deviation of the cost during the scaling. The deviation is an important indicator on how the cost might vary and can help to better plan a system. A low

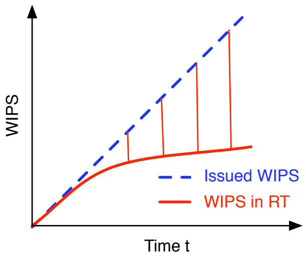

Figure 2: Scalability

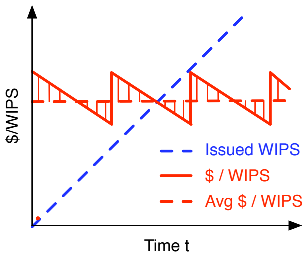

Figure 3: Cost

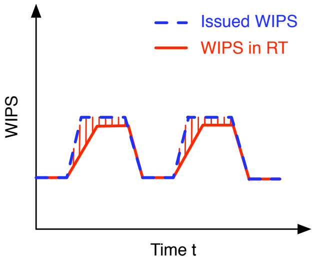

Figure 4: Peaks

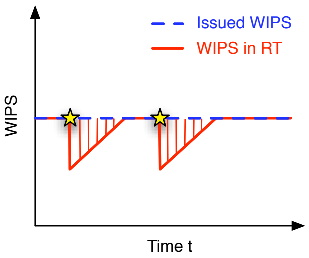

Figure 5: Fault tolerance

value indicates perfect fine-grained utility pricing whereas a bigger value corresponds more to a traditional non-cloud scenario.

Peaks. Next to the possibility to scale, another important property of the cloud is to adapt to peak loads. That is, the focus is not only on the scale-up but also on the scale-down after a peak. To measure the behaviour, we suggest again to vary the issued WIPS over the time as shown in Figure 4. In contrast to the scalability test before, we now assume a certain base load which is only increased for a short period of time.

As a result, the ratio between *WIPS in RT* and *Issued WIPS* can be used to reflect the adaptability to peak loads. A value of 1 indicates perfect absorbing of peaks, whereas a smaller value indicates higher adaptation times. Additionally, as also shown in the Figure, the cost and cost standard deviation have to be measured. The smaller and the more stable the cost the better suited is the system for adapting to peak-loads.

This metric might be influenced by the base load and the elevation of the peak. A too high base load might already push the system to the scalability limit. The different load factors might also result in different average costs. Furthermore, a slow increase in the load is in general better absorbable than fast and sudden increases. Thus, we suggest using several variations of the base load and peaks which are pre-defined by the benchmark. This would also allow for an interesting variation for the peak-metric. Hence it would be possible to report the maximum elevation until the ratio, as defined before, becomes smaller than 1. Thus, the elevation factor indicates the maximum load increase which can be absorbed without user notice.

Fault tolerance. The final metric concerns the failure behaviour of the system. All cloud service implementations that we are aware of base their main infrastructure on commodity hardware. In such deployments, hardware failures are common and not the exception. Hence, a cloud benchmark should not only treat the best case, but also cover how the system behaves in the presence of failures.

The first issue when creating a failure metric is defining what has to fail. We suggest that a certain percentage of the resources used for the application is shut down - regardless if it is a storage service or an EC2 instance. If shutting down the resource is not possible in a fine-granular way, the next resource lot size has to be used (e.g. a complete EC2 instance). Figure 5 indicates this induced failures by the stars. As most cloud computing resources are self-healing, the resources are automatically replaced. Hence, the shutdown of resources can be repeated several times.

Similar to the experiments before, the ratio between *WIPS in RT* and *Issued WIPS* is calculated. A ratio of 1 indicates that the system is completely reliable and can deal with the induced failures in a perfect manner.

Obviously, the failure metric is influenced by the percentage of failures. Choosing a representative percentage or varying the scenario are both valid solutions and open for discussion. Again, alternatively to the ratio it might also be possible to report the maximum percentage of failures without a drop in the ratio. In contrast to the previous metrics, the failure behaviour is most likely only reportable by cloud providers and not cloud users.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we wanted to start an outstanding discussion about the question why existing benchmarks are not adequate for the cloud. Thus, we first showed some arguments, why the existing TPC-benchmarks fall short when analyzing the elastic cloud services. Moreover, we presented a list of requirements for a new cloud benchmark and discussed some initial ideas towards such a benchmark (including some new metrics).

As a part of our future work we want to detail our initial ideas and come up with a more detailed the benchmark specification. Moreover, we plan to implement a first version of such a new cloud benchmark that is based on our ideas and apply it to different cloud architectures.

6. REFERENCES

[1] 28msec, Inc. Sausalito, Apr. 2009. http://www.28msec.com/.

[2] Apache. HBase, Apr. 2009. http://hadoop.apache.org/hbase/.

[3] M. Armbrust et al. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, 2009.

[4] M. Brantner, D. Florescu, D. A. Graf, D. Kossmann, and T. Kraska. Building a database on S3. In *Proc. of SIGMOD*, 2008.

[5] E. A. Brewer. Towards robust distributed systems. In *Proc. of PODC*, page 7, 2000.

[6] M. Cafarella et al. Data management projects at Google. *ACM SIGMOD Record*, 37(1):34–38, 2008.

[7] J.-D. Cryans, A. April, and A. Abran. Criteria to Compare Cloud Computing with Current Database Technology. In *IWSM/Metrikon/Mensura '08: Proceedings of the International Conferences on Software Process and Product Measurement*, pages 114–126, 2008.

[8] Facebook. Cassandra - A distributed structured storage system, Apr. 2009. http://cwiki.apache.org/confluence/display/CSDR/Index.

[9] S. Garfinkel. An evaluation of Amazon's grid computing services: EC2, S3 and SQS, 2007.

[10] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[11] B. Hayes. Cloud computing. *Commun. ACM*, 51(7):9–11, 2008.

[12] Heroku. Heroku - Instant Ruby Platform, Apr. 2009. http://heroku.com/.

[13] TPC. TPC-W Benchmark 1.8. http://www.tpc.org/tpcw/, 2002.

[14] TPC. TPC-H Benchmark 2.8. http://www.tpc.org/tpch/, 2008.

[15] TPC. TPC-C Benchmark 5.10.1. http://www.tpc.org/tpch/, 2009.

[16] W. Vogels. Data access patterns in the Amazon.com technology platform. In *Proc. of VLDB*, page 1, Sep 2007.