Datenbank Spektrum (2010) 10: 121–129
DOI 10.1007/s13222-010-0033-3

SCHWERPUNKTBEITRAG

Data Management in the Cloud: Promises, State-of-the-art, and Open Questions

Donald Kossmann · Tim Kraska

Accepted: 12 October 2010 / Published online: 6 November 2010

Abstract Cloud Computing has the potential to significantly change the IT world. It promises dramatic reductions in cost and time-to-market. This paper gives an introduction to cloud computing, thereby stating how cloud computing tries to fulfill these promises. Furthermore, this paper studies the state-of-the-art of cloud computing platforms and answers the question of how well the current generation of systems meets these promises. Based on that analysis, this paper states several fundamental questions that need to be addressed when designing a cloud computing platform. The focus of the paper is on database workloads; more specifically, on online transaction processing workloads (OLTP) in public clouds.

1 Introduction

The IT requirements of most organizations are simple. First, they want to collect all data that is available. This data includes external data from, e.g., the public Web, and internal data that an organization has collected from their customers, their employees, their products, etc. A second wish is to build services on all the available data. For example, an organization could wish to analyze the data in order to optimize its processes, or an organization could be interested in developing new pricing models based on customer preferences. Most organizations would like to adapt their services frequently and bring them into production as soon as possible. The services should be allowed to access the entire data pool: There are no silos. Third, organizations have a number of operational requirements. For instance, services must never fail. Figure 1 depicts the resulting IT landscape with a layer of services and a layer of data management.

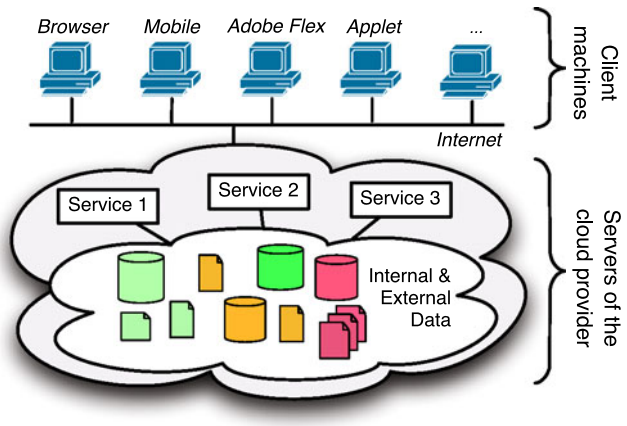


Fig. 1 Data services in the cloud

Guess what? It is possible to achieve all these requirements. The question is no longer whether it is possible to implement a specific service or to collect a particular data set; the real question is whether it is *worthwhile*. Putting it differently, when introducing a new service, organizations ask the following questions: How much will it cost? How long will it take to get it into production? What kind of expertise do I need to develop it and operate it?

Cloud computing promises to lower the bar for new services to become worthwhile by making things cheaper, faster, and automating tasks that traditionally had to be carried out by experts. This way, cloud computing helps optimizing the long tail of processes which are currently not

D. Kossmann
Systems Group, ETH Zurich, 8092 Zurich, Switzerland
e-mail: donaldk@ethz.ch

T. Kraska (✉)
Dept. of Computer Science, UC Berkeley, Berkeley, CA, USA
e-mail: kraska@cs.berkeley.edu

supported by IT and helps to reduce the cost of those processes that indeed are already automated. The purpose of this paper is to list the promises of cloud computing in more detail, study the state of the art, and analyze what needs to be done so that cloud computing fulfills its promises.

Cloud computing technology is typically classified along three dimensions:

- *Deployment Type*: The alternatives are *public clouds* which offer their services to a general audience and *private clouds* which are operated by an organization for its internal purposes.
- *Service Type*: The alternatives are *infrastructure services* (*IaaS*) which provide basic computing resources (CPU cycles, storage, and network bandwidth), *platform services* (*PaaS*) which provide a framework to develop and deploy new, custom services, and *software services* (*SaaS*) which provide a fully-fledged, canned application.
- *Supported Workloads*: The typical workloads that organizations would like to deploy on a cloud are *online transaction processing* (*OLTP*), *analytics* (*OLAP*), *complex event processing* (*CEP*), and *testing*.

While this paper tries to be as general as possible, the focus of this paper is on *public clouds* that provide *platform services* (*PaaS*) (e.g., databases, queues, and application servers), and support *OLTP workloads*.

According to its goals, the rest of this paper is structured as follows: Sect. 2 revisits the basic principles of cloud computing and its most important promises. Section 3 lists the most important offerings on cloud computing and summarizes the main results of a recent study [11]. Section 4 discusses some of the critical open questions with regard to designing a data management system in the cloud. Section 5 contains conclusions. This paper is based on a recent keynote presentation at the IEEE Data Engineering Conference [10]. The slides of that talk are available at the conference web site, at the ETH Systems Group web site (http://systems.ethz.ch), and upon request from the authors.

2 Promises

This section discusses why cloud computing can help to reduce cost and time-to-market. Before doing so, however, the most important principles of cloud computing are explained.

2.1 Principles of Cloud Computing

Cloud computing is based on three important principles that depend on each other and can only provide additional value if implemented in concert:

1. *Automation*: Mundane task of maintaining an IT infrastructure such as starting a machine, stopping a machine, installing software, backups, etc. are automated. More concretely, a cloud computing provider offers a REST or Web Service API for executing such tasks in an automated way.
2. *Virtualization*: This principle mandates that no piece of software (neither data nor program code that defines services) is ever bound to any hardware resources. Virtualization makes it possible to provision hardware resources flexibly to data and services. It is, therefore, a requirement for the pay-as-you-go paradigm.
3. *Pay-as-you-go pricing model*: Users only pay for hardware and software resources that they consume. That is, consumption of resources is metered and the unit of metering is typically fine-grained. For instance, storage and network bandwidth consumption are typically metered in the granularity of GBs. CPU consumption is typically priced per hour.[1] To make this pay-as-you-go model work, the provided services must be *elastic*. That is, users must be able to release resources if they are no longer needed and users must be able to provision additional resources if required. Ideally, a PaaS offering implements elasticity automatically. So the platform automatically grows and shrinks the provisioned resources for a specific service based on the demand for that service.

The remainder of this section shows how these principles in concert—if implemented correctly—can help to reduce cost and time-to-market.

2.2 Cost

There are a number of reasons why C*F*Os are getting interested in cloud computing. The first reason is a direct consequence of the *pay-as-you-go model*. As a result, organizations turn capital expenses into operational expenses. In other words, organizations avoid investments that may never pay back if the new service is not a success.

Even for successful services, cloud computing can have several cost advantages: First, the *utilization* of the hardware resources can be improved. This effect is a direct consequence of *virtualization*. If services are flexibly assigned to hardware resources, then several services with little demands can be packed into the same machine. If the popularity and workload of one of these services increases, then that service can be moved to another machine. In all, bin packing can be used in order to minimize the number of machines that are active at any moment in time. Correspond-

[1] Amazon Elastic Compute # (Amazon EC2). http://aws.amazon.com/ec2/, 2010.

ingly, cloud computing is often considered an enabling technology for *green computing* (aka energy-efficient computing). In related work, the cost savings of cloud computing have been referred to as *statistical multi-plexing of computing resources* [1].

Further cost reductions can be achieved by an economy of scale. Cloud computing providers can purchase hardware cheaper because of volume effects. Furthermore, cloud computing providers can operate data centers at locations with cheap energy, another direct consequence of virtualization. Finally, it is worth to employ super-engineers to operate and optimize these data centers. Most organizations cannot afford (nor attract) such engineers because the optimization potential is not big enough. In a PaaS environment, it is possible to continuously fine-tune the infrastructure and the platform in order to achieve further cost reductions. Such opportunities typically do not arise in a traditional software deployment model because upgrades to new releases of platform components (e.g., databases) require a huge effort.

Finally, cloud computing promises to reduce the cost to *avoid failures* and the cost of *having failures*. Cloud computing is typically implemented on clusters with cheap commodity hardware instead of expensive mainframes. Failures inside this infrastructure are the normal case and not an exception. In order to provide highly reliable services on this unreliable infrastructure, cloud services have to be designed to automatically deal with all kinds of failures. Thus, cloud services typically use home-grown distributed fabrics based on recent results of distributed computing research in order to detect, tolerate and/or automatically repair various kinds of failures [5, 7]. This way, the cost of avoiding failures is reduced as cheaper hardware and fewer stand-by engineers on weekends and at night are required. The cost of having failures is reduced by minimizing the effect of any kind of failure on the system.

2.3 Time to Market

Arguably, the biggest advantage of cloud computing is the potentially faster time-to-market. In many organizations, the savings in IT infrastructure cost are negligible compared to the lost opportunity of being fast. Cloud computing can reduce time-to-market because it eliminates several steps in the process of developing and deploying a service. Most importantly, up-front *hardware provisioning* is not necessary if cloud computing technology is adopted. Again, as a result of virtualization and the pay-as-you-go paradigm, organizations should not think about the hardware resources that a service may consume once it is deployed. Instead, the required hardware resources should be provisioned dynamically based on the real (rather than estimated) workload. Estimating the workload generated by a service and provisioning hardware accordingly is an expensive and time-consuming task. This task can only be carried out *after* the service has been implemented because it is based on benchmarking. Therefore, removing this task will immediately impact the time-to-market.

In general, it is difficult to measure time-to-market of a software project. In particular, it is difficult to estimate the savings in time-to-market of a particular cloud computing product. However, it is possible to benchmark the *elasticity* of a cloud computing product. If a cloud computing product is indeed fully elastic, then hardware provisioning is not an issue and time-to-market can be reduced.

3 State of the Art

While the promises of cloud computing are compelling, it is not at all clear that all cloud computing providers deliver on these promises. This section outlines the results of a recent study that assessed the state-of-the-art [11]. In a nutshell, that study showed that the differences in cost and elasticity of the different providers are immense. The next section tries to look behind the curtain and lists crucial design questions that cloud providers must face.

3.1 Players

There are a number of companies that provide cloud services or cloud-enabled services at various levels and for different kinds of clouds. Arguably, the three most prominent companies that have PaaS offerings in a public cloud are Amazon, Google, and Microsoft. As mentioned in the introduction, this work focuses on this kind of cloud services. This subsection describes the offerings of these companies at a high level. The market is highly dynamic and the services itself and many of their details are subject to constant change.

3.1.1 Amazon

Amazon offers a suite of IaaS and PaaS services that can be used to deploy data-intensive services in the cloud as part of Amazon Web Services, AWS for short.[2] Among the most famous IaaS services are S3 and EBS for data storage and EC2 for the rental of virtual machines. Amazon's most prominent PaaS services are RDS and SimpleDB. Amazon RDS is a so-called *database as a service*. That is, RDS allows users to create a relational database and execute SQL statements (schema, data manipulation, and queries) on such databases using a REST or Web Service interface. The service requires managing the database instance directly. Thus, users start and stop database instances and control the *size* of the CPU and the memory of the instance that handles SQL requests to the database. Amazon charges per CPU hour that

[2] Amazon Web Services. http://aws.amazon.com/, 2009.

the database is online, depending on the selected CPU and memory of the database server. In addition, prices may vary by region: In 2010, using a data center in the USA has been cheaper than using a data center in Europe.

SimpleDB is an alternative to RDS. As opposed to RDS, SimpleDB does not support fully-fledged SQL. Instead, SimpleDB provides a simplified query interface that allows executing range queries, but no joins or aggregates. If an application programmer wishes to execute those kinds of queries on top of SimpleDB, custom code must be written as part of their application. SimpleDB has a slightly more flexible data model than RDS which is based entirely on the SQL (relational) data model: In SimpleDB, records of the same *domain* can have different formats. Furthermore, the pricing model of SimpleDB is different: In SimpleDB, users pay for the consumed CPU hours and traffic while processing queries.

Both SimpleDB and RDS can be used in order to deploy (OLTP) services. Neither of them, however, is sufficient because they merely provide the database tier. In order to deploy the application logic and a Web server, Amazon provides the EC2 service. EC2 allows provisioning virtual machines in the Amazon cloud using, again, a REST or Web Service interface. These virtual machines can be configured to run Linux or Windows and can be used to install any kind of software (e.g., application servers). It is possible to select a data center for the EC2 machines and, of course, it makes sense to collocate EC2 machines and database services (i.e., SimpleDB or RDS) in the same data center. EC2 prices, again, depend on the time (measured in the granularity of hours) that an EC2 virtual machine is rented, the size of the CPU, and possibly the location of the data center. Furthermore, network bandwidth to communicate with EC2 machines from the outside is charged. The total cost of running a service in the Amazon cloud is the cost to run EC2 machines plus the cost to run SimpleDB and/or RDS plus the outgoing and ingoing network traffic of the services into and out off the Amazon cloud.

3.1.2 Google

Google has a dedicated PaaS service called AppEngine. AppEngine allows to develop and deploy data-intensive services written in Java or Python. Furthermore, AppEngine provides a service to host a relational database and enables embedding queries and update statements to that database into the Java or Python code. As a database language, AppEngine supports GQL (Google Query Language) which is an SQL dialect. GQL is a dramatic simplification of SQL, somewhere in between real SQL and the queries supported by SimpleDB. For better or worse, AppEngine does not provide much flexibility and control on how the application and the database is deployed in the Google cloud. All configuration is done automatically. The pricing model is a true pay-as-you-go model: That is, if a service is (virtually) inactive, no cost is incurred. (Only a marginal monthly fee to keep the data of the database consistently is charged.) For active services, Google charges network traffic and the CPU hours consumed for processing requests.

3.1.3 Microsoft

Recently, Microsoft has also started to offer a suite of IaaS and PaaS services to deploy data-intensive services in their cloud. This suite of services is branded *Microsoft Azure*. The services and the whole approach resemble those offered by Amazon AWS and it can be expected that the similarities will become even bigger in the near future. In order to host data-intensive services, for instance, Azure provides two particular services: (a) Windows Azure which plays the same role as EC2 and allows running application logic in a virtual machine; (b) SQL Azure which corresponds to RDS and supports a relational database. The pricing model of Windows Azure is the same as of EC2. The pricing model of SQL Azure involves a flat monthly fee that depends on the size of the database and includes the cost of processing queries (independent of the query workload).

3.2 Results

The results of a comprehensive performance study comparing the current offerings from Amazon, Google, and Microsoft for OLTP workloads have been presented in detail in [11]. In the remainder of this section, these results are briefly summarized in order to motivate the discussion of the next section. We believe that these results are representative for the state-of-the-art in PaaS cloud computing in the year 2010 as Amazon, Google, and Microsoft are currently the most prominent players in the market. We present our findings with regard to cost and elasticity. These two criteria are crucial to implement the most important promises of cloud computing. (As stated in Sect. 2.3, elasticity is important in order to reduce the time-to-market.) All experiments were carried out with the TPC-W benchmark.[3] Again, more details of the experimental set-up and explanations of the effects and results can be found in [11].

3.3 Cost

Table 1 shows the cost per Web Interaction of the various ways to deploy the TPC-W on the platforms provided by Amazon, Google, and Microsoft. A Web Interaction of the TPC-W benchmark models the click of a user in an online bookstore (e.g., searching for a book, putting a book into a

[3]Transaction Processing Performance Council. TPC-W 1.8. http://www.tpc.org/tpcw/, 2002.

Table 1 Cost per Web interaction [m$], Vary EB

	Emulated Browsers (EB)						
	1	10	100	500	1000	3000	9000
Amazon RDS	1.211	0.126	0.032	0.008	*0.006*	*0.005*	–
Amazon SimpleDB	0.384	0.073	0.042	0.039	0.037	–	–
Google AppEngine	*0.002*	*0.018*	0.026	0.028	–	–	–
MS SQL Azure	0.775	0.084	0.023	*0.006*	*0.006*	*0.005*	*0.005*

shopping cart, etc.). The TPC-W benchmark defines several mixes of such Web Interactions and Table 1 shows the results for the so-called *Ordering Mix*. Furthermore, Table 1 shows the cost per Web Interaction for a varying number of emulated browsers (EB). An emulated browser simulates a user. So, the column for 1000 EBs shows the results for 1000 concurrent users. Again, the TPC-W benchmark defines the thinking time of such an (EB) user.

Intuitively, the cost of a Web Interaction should not depend on the number of concurrent users. If you buy milk in a supermarket, then you would not expect the price per gallon to depend on the number of other people who are in the same or in another supermarket. You would expect to pay the price that is displayed at the cooling rack. As shown in Table 1, unfortunately, all providers of the state-of-the art have prices which vary depending on the load. Even worse, the prices are completely unpredictable because all cloud providers sell CPU hours or network bandwidth, but not "Web Interactions" (or requests) which is what really matters to businesses. Surprisingly, the different platforms show different behavior. While for Google AppEngine the prices per Web Interaction increase with the number of users, the prices for Amazon and Microsoft decrease with the number of users. This is due to the fact, that Google AppEngine has an initial free quota whereas Amazon and Microsoft have high starting costs (e.g., $10–$100 per month for SQL Azure) which only amortize with bigger workloads.

The most striking observation that can be drawn from Table 1 is that the differences in cost are huge. There are no differences in quality here: A TPC-W Web Interaction is a TPC-W Web Interaction, regardless of whether it is served by the Amazon or, say, the Google cloud. (All providers have roughly comparable SLAs and response times—so, there really is no difference in the quality of a Web Interaction here.) Continuing the "milk" example, you would expect that milk is more expensive in a boutique shop on Fifth Avenue, New York, than in a supermarket in, say, Kansas City. However, you would not expect the differences in price to be three orders of magnitude. As Table 1 shows for the cloud offerings, the differences in price can be several orders of magnitude between different providers. Even worse, the differences in price can be several orders of magnitude even with the same provider, depending in this case on the number of concurrent users, a parameter which is typically not known in advance.

Table 2 Maximum load sustained [EBs]

	Max. EB
Amazon RDS	3750
Amazon SimpleDB	1000
Google AppEngine	500
MS SQL Azure	*9000*

The big question is not answered by Table 1: Does cloud computing reduce cost? Unfortunately, there is no simple answer to this question (otherwise, we would try to give it) because it is difficult to find the right baseline. Unfortunately, the official TPC Web site no longer lists results for the TPC-W benchmark on traditional IT infrastructures so it is not possible to infer a baseline from there. The huge variances shown in Table 1 indicate, however, that cloud computing can reduce cost if done right, but it can also significantly increase cost if not done right.

3.4 Elasticity

As stated in Sect. 2.3, one promise of cloud computing is to reduce time-to-market for IT projects. As explained in that section, a cloud computing solution must be elastic in order to fulfill that promise. That is, the provider must be able to adapt quickly to workload changes. Benchmarking elasticity is not trivial and we are currently working on a comprehensive benchmark for this purpose. (Initial ideas are presented in [3].) Table 2 shows the results of a simple experiment that we conducted in order to see at what point a cloud provider hits its limits. Obviously, a fully elastic service should (virtually) never hit its limits. Unfortunately, even with a modest (university) budget it was possible to reach the limits of a single deployment of the TPC-W in the Amazon and Google clouds.

The only deployment of the TPC-W benchmark which was able to sustain our maximum load of 9000 EBs (i.e., simulated concurrent users of a book store) was MS SQL Azure. Amazon RDS drops out at 3750 EBs. The reason why MS SQL Azure out-scales Amazon RDS although they

use a similar architecture (see Sect. 4.1) is, that Amazon RDS uses a less powerful machine for the primary copy. As a matter of fact, MS SQL Azure also has an inherent scaling limit of the master copy, which was simply not reached in this experiment. Amazon SimpleDB was able to handle a load of 1000 EBs before it started to drop write requests. The worst performance in this experiment had Google AppEngine with only 500 EBs. The main reasoning behind it is, that we tried to follow the TPC-W specification as close as possible including the consistency requirements. Google AppEngine offers transactions so we used it to implement the consistency requirement of TPC-W. Only afterwards, we were informed, that Google AppEngine does not implement concurrent transactions. Thus, Google AppEngine was executing one transaction at a time. We assume, that without using the offered transaction guarantees, Google AppEngine would have been able to scale further.

4 Big Questions

The previous two sections described the promises of cloud computing and analyzed the state-of-the-art. Our conclusion is that cloud computing does not yet live up to its promises. One reason may be that the products are not yet mature. That is reasonable because most offerings have been launched only recently and we expect all products to make quantum leaps in cost and performance in the near future. There seem to be, however, also more fundamental design issues on how to build a PaaS for cloud computing. This section summarizes the discussions on seven such design questions that we have been debating with people in the research community and industry.

4.1 How to Partition the Data?

Figure 2a shows the reference architecture of Web-based data services. (HTTP)-requests from clients are handled by a Web server which passes them to an application server. The application server executes the application logic written in a programming language like Java or C#. The application code includes calls to the database in order to read and write objects; typically, those calls are implemented using SQL. Calls to the database are handled by a database server. The database server evaluates the SQL queries and synchronizes concurrent calls to the same data. Finally, the data is stored persistently by a storage system. The storage system could be a Storage Area Network (SAN) or disks which are attached locally to the machine that runs on the database server.

The big question is how to map the components of the architecture of Fig. 2a to (virtual) machines in the cloud. As will become clear in the remainder of this subsection, the answer to this question implies how data is partitioned, how the consistency of the data is maintained, and how load balancing is carried out. Figures 2b and 2c sketch two fundamentally different approaches to answer these questions.

Figure 2b sketches the approach that was pioneered by Salesforce. The idea is to partition the data by *tenant*. A tenant is a customer of the Force.com cloud. Correspondingly, all requests will be routed by tenant. That is, all requests of the same customer are handled by the same Web server, application server, and database server. The Web and app servers of several tenants can be hosted by the same machine. Likewise, the same machine can serve the database of many tenants.

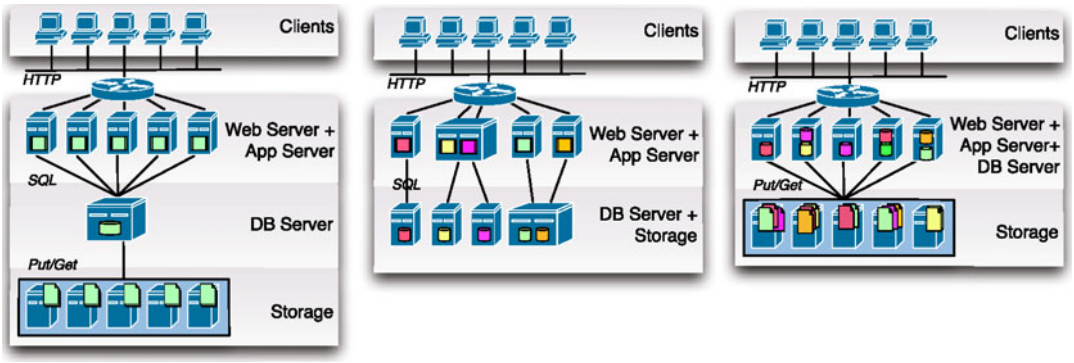
The big advantage of the Salesforce approach is that off-the-shelf Web, app, and database servers can be used. Furthermore, several small customers can share the same machine so that a high utilization can be achieved, one of the most important goals of cloud computing (Sect. 2). On the negative side, Salesforce needs to apply complex optimizations in order to determine which set of tenants should be hosted by which machine. Essentially, this task involves solving a dynamic bin-packing problem. If the load of a tenant grows, it may become necessary to move it (or another co-located tenant) to a different machine in order to meet the SLAs.

The architecture of Fig. 2b has also been adopted by Microsoft as part of its SQL Azure service. In SQL Azure, applications define a logical unit of consistency. This logical unit of consistency represents a set of tables that are managed by the same database server. All requests to data of such a logical unit of consistency are synchronized by that database server. Again, SQL Azure hosts several logical units of consistency with a single machine, thereby solving a similar bin-packing problem as Salesforce. In order to limit the cost of moving logical units of consistency between machines, SQL Azure limits the size of a logical unit of consistency; currently, the limit is 50 GB. If a service requires access to a larger data pool, the application code of that service must federate between several logical units of consistency. Currently, Microsoft provides no support for this federation task.

The biggest downside of the approach is, that the architecture requires finding a partitioning scheme, and that the load of every partition fits entirely on at least one server. In the light of many new social applications data becomes increasingly more connected. Thus, finding a good partitioning scheme becomes increasing more complicated.

Figure 2c depicts the approach taken by Google, 28 ms, and several other start-ups. Many of the proponents of the so-called NoSQL movement[4] seem to have adopted this approach. The idea is to pool all the data and all computing

[4]S. Edlich, The List of No SQL databases. http://nosql-database.org/, 2010.



(a) Reference (b) Partitioning (c) Load

Fig. 2 Architectures

resources of a data center. That is, the data is stored in a distributed storage system such as a key-value store or a distributed database system. Each incoming request may be handled by a different (virtual) machine; a virtual machine that is currently not busy. That virtual machine will load the application code to handle the request from the distributed storage system, interpret that application code thereby reading and updating data objects from the distributed storage system. That is, all these virtual machines are stateless.

The big advantage of this approach is that it avoids any kind of data silos: Any service can virtually access all the data stored in the cloud as shown in Fig. 1. A second advantage is that failures are easy to handle: Since the virtual machines are stateless, they can fail anytime without data loss; only the requests that were currently processed by that machine are lost. Fault-tolerance at the storage level can be implemented using data replication. A third advantage is that a high utilization can be achieved without the need to solve a complex bin-packing problem. On the negative side, concurrent accesses from different virtual machines to the same data object need to be synchronized. Furthermore, this approach involves re-implementing essentially the whole system's stack. Doing so brings the opportunity to simplify the stack and merge several layers into a single tier as shown in Fig. 2c and for example implemented in [4].

The final verdict which of the two approaches, Fig. 2b vs. Fig. 2c, to adopt for data services in the cloud, has not been made. The experimental results presented in [11] indicate that the approach of Fig. 2c is more cost-effective if many small services need to be hosted. On the other hand, large-scale applications with strong consistency requirements can only be handled using the approach of Fig. 2b given the current state-of-the-art.

4.2 Consistency vs. Availability?

A second open question that needs to be addressed in both approaches discussed in the previous subsection is, how to deal with the infamous CAP theorem [8]. This theorem specifies that it is impossible to achieve strong consistency of data (C), 100 percent availability (A), and resilience to network partitioning (P). It is possible to achieve two of these properties, but not all three.

Since network partitioning is a fact of life, two schools have formed. The first school propagates that strong consistency is crucial and tries to maximize availability under this constraint. The second school, on the contrary, prioritizes availability and tries to maximize consistency. Both camps have strong arguments: Strong consistency (i.e., ACID transactions) makes it easier to build applications and to operate data services. Without strong consistency, complex application logic needs to be implemented that detects and compensates for data inconsistencies. Often, such data repairs involve expensive manual work. Prioritizing availability is also justified because businesses loose revenue whenever their services are not available.

Again, there is no clear answer to the question "consistency" or "availability". In practice, the right answer depends on the application and business scenario. As a result, various levels of consistency have been defined [14, 17, 18] in order to give application developers a way to trade consistency for increased availability. A more recent line of work has studied protocols to trade consistency for cost [4, 12].

4.3 Control Architecture?

Together with the question of consistency and partitioning comes the question of the control architecture. Most current distributed database systems use master/slave architectures which make it easy to implement strong consistency guarantees. In such a scenario, the master coordinates all writes and propagates the decisions to the slaves. Depending on the configuration, either the master is responsible for all reads and writes (e.g., MS SQL Azure uses this model) or some/all of the reads are offloaded to the slaves. The advantage of the master/slave architecture is the simplicity of the design. On the downside, the master is a potential bottleneck in the system. Further, this kind of system reduces the availability because it requires always a connection to the master for writes at all times.

In multi-master architectures, a set of masters share the responsibility by acting on either distinct ranges of the data or assigned individual sets of items. For example, Yahoo PNUTS [6] uses a multi-master architecture in which the mastership for every record can be individually assigned to one of the geographically distributed nodes in the system. Obviously, for such a model multi-record transaction guarantees are harder and more expensive to achieve than in the case of a single master architecture. However, failures only render parts of the systems unavailable for writes.

An extreme form of multi-master architectures is fully distributed control, often using distributed hash tables such as Chord [16] or Pastry [15] for routing request. Here, the ownership of records seamlessly migrates between servers in the case of failures, making the approach well-suited for high availability. However, implementing stronger levels of consistency implies higher overhead in this architecture. Amazon Dynamo [7] is an example for a system using this control architecture.

Although, it is possible to cover almost the full spectrum of consistency vs. availability for the different partitioning schemes, some control architectures are more suited than others for certain system requirements.

4.4 Other Open Questions

There are a number of other open questions which have been addressed in the literature and which are relevant for the right design of a cloud infrastructure:

How to store the Data? As shown in Fig. 2, the storage system is a critical component of the systems stack. There are three open questions evolving around this layer:

- *What is the right API of the storage system*? Traditionally, a storage system (e.g., a disk or a SAN) has a simple put/get interface. Data items are identified by keys (e.g., a URL or a block address) and retrieved in the granularity of blocks (e.g., chunks of 512 bytes). With current hardware trends, it might make sense to implement richer APIs. For instance, it might become economically viable to push down predicates and richer query processing semantics into the storage system. This idea is old and has, for instance, been explored in the context of idisks [9]. Lately, the same idea has been implemented in the context of main-memory databases, e.g., [19].
- *What is the right data model*? Next to the well-known relational model, key-value stores have become increasingly popular. As the name implies, the simplest form is an (un-)ordered set of key/value pairs. Dynamo [7] and Amazon S3[5] are the most prominent examples using this model. A natural extension to this model is to associate more than one value to a key. This model is for example supported by BigTable [5], PNUTS [6] and Amazon SimpleDB.[6] As the biggest remaining difference to the relational model, the key/values are not required to be of the same relation and can be more flexibly defined.

  However, recently more and more exotic data models have been used in the various systems. Ranging from the possibility to store objects (e.g, MongoDB[7]) up to three-dimensional structures (e.g., Cassandra[8]). How to generalize the advantages of the different new models is still an open question.
- *What is the right storage media*? There have been a great deal of advancements in storage technology, the latest development being phase-changing memory [13]. The traditional approach relies heavily on hard disks as part of a storage hierarchy. Finding the right performance/cost trade-offs is still an open question. Furthermore, it is unclear how to evolve a storage cloud with evolving technology.

What is the right programming language? Another heated debate concerns the right programming language for developing data services in the cloud. Again, there are two camps. The first camp insists that the choice of the programming language should be made independent of deployment considerations. That is, the same programs should run in the cloud as in traditional IT infrastructures. The other camp argues that new programming paradigms are needed in order to exploit the promises of the cloud. This discussion is tightly coupled to the discussion on the other open technical questions. Existing programming languages and legacy applications can only be cloud-enabled if the approach of Fig. 2b is chosen. Only in that approach can existing application servers be leveraged in the cloud. The proponents of the approach of Fig. 2c argue for other more constrained programming models. As mentioned in Sect. 4.1, the NoSQL movement belongs to this camp. Noticeably, Google AppEngine only supports subsets of Java and SQL; again, trading programming comfort for cost efficiency. PIQL is a recent approach to restrict SQL in order to achieve predictable performance [2].

5 Conclusion

Although cloud computing is often considered as a hype, most people also agree that it is here to stay due to the as-

[5] Amazon, Simple Storage Service (S3). http://aws.amazon.com/s3/, 2009.

[6] Amazon, SimpleDB Developer Guide (API Version 2009-04-15). http://docs.amazonwebservices.com/AmazonSimpleDB/, 2009.

[7] 10gen, MongoDB. http://www.mongodb.org, 2010.

[8] Facebook, Cassandra. http://incubator.apache.org/cassandra/, 2009.

sociated economical benefits. Nevertheless, none of the currently available solutions covers all of the promises of the cloud yet. At the same time, more and more systems appear on the market with an increasing variety of approaches to achieve the promises. It remains to be seen, which of the possible architectures, programming languages, data models will survive over time and if the architectures will consolidate to one reference architecture as happened for the relational database system.

Acknowledgements This work would not have been possible without the author's collaborators at 28 ms and the ETH Systems Group. Specifically, the authors would like to thank (in alphabetical order) Gustavo Alonso, Roger Bamford, Carsten Binnig, Lukas Blunschi, Irina Botan, Matthias Brantner, William Candillion, Dietmar Fauser, Peter Fischer, Daniela Florescu, Ghislain Fourny, Michael Franklin, Georgios Giannikis, David Graf, Laura Haas, Martin Hentschel, Stefan Hildenbrand, Paul Hofmann, Claudio Jossen, Dennis Knochenwefel, Alexander Kreutz, Francois Laburthe, Simon Loesing, Stephan Merkli, Renée Miller, Raman Mittal, Gabriel Petrovay, Flavio Pfaffhauser, Timothy Roscoe, Peter Schnorf, Kurt Stockinger, Nesime Tatbul, Beth Trushkowsky, Philipp Unterbrunner, and Roger Weber.

References

1. Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson DA, Rabkin A, Stoica I, Zaharia M (2009) Above the clouds: a Berkeley view of cloud computing. Technical report UCB/EECS-2009-28, EECS Department. University of California, Berkeley
2. Armbrust M, Tu S, Fox A, Franklin MJ, Patterson DA, Lanham N, Trushkowsky B, Trutna J (2010) PIQL: a performance insightful query language. In: SIGMOD conference, pp 1207–1210
3. Binnig C, Kossmann D, Kraska T, Loesing S (2009) How is the weather tomorrow? Towards a benchmark for the cloud. In: Proc. of DBTest workshop (SIGMOD), 2009
4. Brantner M, Florescu D, Graf D, Kossmann D, Kraska T (2008) Building a database on S3. In: Proc. of SIGMOD, pp 251–264
5. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE (2006) Bigtable: a distributed storage system for structured data. In: Proceedings of OSDI
6. Cooper B, Ramakrishnan R, Srivastava U, Silberstein A, Bohannon P, Jacobsen H, Puz N, Weaver D, Yerneni R (2008) PNUTS: Yahoo!'s hosted data serving platform. In: Proceedings of VLDB
7. DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W (2007) Dynamo: Amazon's highly available key-value store. In: Proc. of SOSP, pp 205–220
8. Gilbert S, Lynch N (2002) Brewer's conjecture and the feasibility of consistent available partition-tolerant Web services. In: ACM SIGACT news
9. Keeton K, Patterson DA, Hellerstein JM (1998) A case for intelligent disks (IDISKs). SIGMOD Rec 27(3):42–52
10. Kossmann D (2010) How new is the cloud. In: ICDE
11. Kossmann D, Kraska T, Loesing S (2010) An evaluation of alternative architectures for transaction processing in the cloud. In: SIGMOD conference, pp 579–590
12. Kraska T, Hentschel M, Alonso G, Kossmann D (2009) Consistency rationing in the cloud: pay only when it matters. In: Proceedings of the VLDB endowment, vol 2
13. Lee BC, Zhou P, Yang J, Zhang Y, Zhao B, Ipek E, Mutlu O, Burger D (2010) Phase-change technology and the future of main memory. IEEE Micro 30(1):143
14. O'Neil PE (1986) The escrow transactional method. ACM Trans Database Syst (TODS) 11(4):405–430
15. Rowstron AIT, Druschel P (2001) Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Proc. of middleware. Springer, Berlin, pp 329–350
16. Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H (2001) Chord: a scalable peer-to-peer lookup service for internet applications. In: Proc. of SIGCOMM. ACM, New York, pp 149–160
17. Tanenbaum AS, Steen MV Distributed systems: principles and paradigms, 2 edn. Prentice Hall, New York (2006)
18. Terry DB, Demers AJ, Petersen K, Spreitzer M, Theimer M, Welch BB (1994) Session guarantees for weakly consistent replicated data. In: PDIS, pp 140–149
19. Unterbrunner P, Giannikis G, Alonso G, Fauser D, Kossmann D (2009) Predictable performance for unpredictable workloads. PVLDB 2(1):706–717