Evaluating DBaaS Offerings

Irina Astrova[1], Arne Koschel[2], Chris Eickemeyer[2], Jan Kersten[2], Norman Offel[2]
[1]*Tallinn University of Technology, Estonia*
[2]*University of Applied Sciences and Arts, Germany*

Abstract

Database-as-a-Service (DBaaS) offerings are increasingly attractive to customers because they provide businesses with databases that can be up and running in a matter of minutes without a lot of staff training. Many DBaaS offerings are relational databases requiring the use of SQL, but some are based on alternative technologies like NoSQL databases. In this paper, we look at both. In particular, we select DBaaS offerings from Amazon and Microsoft (viz., Amazon RDS, Amazon DynamoDB, Azure SQL Database and Azure DocumentDB) and parse out similarities and differences in their features.

1. Introduction

DBaaS (Database-as-a-Service) aims to relieve customers from the burden of setting up and managing databases by their own so that the customers can focus on their core business. Additional benefits of DBaaS include rapid provisioning of potentially endless resources, on-demand scalability, performance tuning and a pay-per-usage pricing model, making DBaaS well-suited for applications that are often idle, can grow rapidly, can encounter unpredictable peaks or need fast deployment [1].

The requirements for DBaaS primarily emerge from two main areas: cloud computing and database management system (DBMS). Cloud computing requires that DBaaS offerings should meet the following requirements:

- **On-demand provisioning of resources:** Changes require minimal interaction with the DBaaS provider.
- **Rapid deployment of services:** Service instances are usable within few minutes.
- **Shared resources:** Customers or tenants share a pool of resources, but each one is isolated.
- **Configurability of services:** The service itself, and only the service, may be reasonably configurable. In terms of DBaaS, the configuration is limited to the features of the provided database functionality. The customer should, e.g., be able to configure access control, performance features or geo-replications.
- **Ubiquitous availability of services:** Each service instance is worldwide available and accessible via the Internet.

DBMS offerings found the second base technology for DBaaS. DBaaS offerings strive to provide database functionality and thus, have to hold the following requirements:

- **Persistence:** DBaaS should offer a durable and permanent data storage.
- **CRUD operations:** DBaaS should offer a means to create, read, update and delete the stored data.

2. Candidates

For our evaluation, we selected two popular DBaaS providers (viz., Amazon and Microsoft), which offer both relational and NoSQL databases. Amazon is the undisputed market place leader in cloud computing coming with Amazon RDS [2] as a relational database and Amazon DynamoDB [3] as a NoSQL database.

We compared this provider with Microsoft, in particular, with its relational database Azure SQL Database [4] and NoSQL database Azure DocumentDB [5]. Microsoft is a long-term competitor of Amazon and with its recent slogan “cloud first” it will become an even more formidable competitor, who starts to enter the market of DBaaS offerings.

3. Evaluation criteria

Modern DBMSs have to meet a lot of requirements to provide applications with means to manage vast amounts of data. Typical requirements are first and foremost performance, availability and scaling but also abstraction for applications and easy administration and configuration. Additional requirements for DBaaS are a means to guarantee privacy and security of the customers’ data, the ease of changing DBaaS providers and the transparency of underlying technologies and hardware.

The feature of DBaaS to outsource the data management is also the point from which most challenges arise. One is the additional overhead of remote access to the stored data. For some applications, this latency can be a major drawback of DBaaS. Another problem is the transparency of the underlying technologies. The main goal of cloud

computing is to accomplish an abstraction of everything underneath the provided service. Therefore, a DBaaS provider needs to find a way to allow an abstracted view of the DBaaS for end users. The interface needs to enable commonly used database languages like SQL, MapReduce or language integrated query interfaces depending of the underlying database management software. Furthermore, since some DBMSs are not specifically designed with a distributed setup in mind, the offered functionality can be only a subset of the capabilities of the underlying database. Customers need to evaluate whether those features suffice for each application. As these are the only ways of direct interaction for end users with the remote database, customers are bound to them and cannot install or demand individual interactions. To be less dependent on the provided interfaces, a standardized DBaaS API (Application Programming Interface) or an interaction model is needed but currently not established.

Every time customers agree to use a service of an external provider, they need to establish a contract that guarantees a minimum SLA (Service Level Agreement). The difference between DBaaS SLA and other services SLA is that it has to cover the performance factors like a throughput, the number of concurrent queries, an available database size and the number of transactions per time unit. In cloud environments, hardware specifications are not interesting for customers as they do not know what runs on top of it and most of the time the resources are shared with other customers. Depending on other virtualized machines on the same hardware, the overall performance of the individual machine or database can vary drastically.

To offer customers a way to really pay what they need in terms of performance, DBaaS providers like Microsoft introduced the measurement of Database Throughput Units (DTUs). DTUs provide a way to describe the relative capacity of a performance level based on a blended measure of CPU (Central Processing Unit), RAM (Random Access Memory), reads, and writes. Other DBaaS providers like Amazon offer similar metrics to abstract the term performance. The benefits for customers are that they can then expect stable performance of their remote database and that they can switch to higher or lower levels as they need. The DBaaS provider will take care of the provisioning, hardware and instances needed to achieve the performance level customers agree on.

For customers, the data they own and work with are invaluable. Therefore, DBaaS providers need to guarantee the privacy of those data. In fact, trust and privacy are what is hindering the success of DBaaS to this day the most. There needs to be a mechanism that ensures that no other customer of the provider and even the provider itself cannot see or deduct any concrete information of the stored data.

Security should also be ensured for the underlying DBaaS. Updates for platforms and database management software should be installed immediately and with no downtime for that DBaaS. When end users cancel the subscription or just delete data from their database, those data should be securely deleted. This means that there should be no way to recover the data either for customers or DBaaS providers.

From the DBaaS providers' point of view, challenges arise from abstraction of the underlying software and hardware and from the customers' need of dynamic adaptability of the performance. Like any service, DBaaS has to present itself in an abstracted manner. End users should be able to configure or access the database functionality without knowledge of the setup of DBaaS on the provider's side. Therefore, DBaaS should present a simple interface for data management and querying. It needs to obscure whether the database is shared or not, or if every database owns a virtual machine.

Yet another major challenge for DBaaS providers is consistency and reliability in large-scale databases. They need to fulfill consistency requirements of their customers but additionally need to be able to adapt to dynamic changes in their performance and allow for a low latency all over the world. Scalability is implemented by partitioning in DBMSs where a database is split in a number of parts, which can be located on different servers. However, as Brewer's theorem states, it is impossible for a distributed database to simultaneously provide more than two out of the following three guarantees: consistency, availability and partition tolerance. Therefore, customers and DBaaS providers need SLAs again.

To summarize, we evaluated the candidates against the following criteria:

- User interface;
- Backup and recovery capabilities;
- Scalability, replication and availability;
- Privacy and security;
- Database language richness;
- Pricing;
- Maintenance;
- Performance.

4. User interface

Amazon RDS: Using the graphical user interface (GUI), end users can create a new database instance or if the instance already exists, different views of that instance. However, when creating a new database instance and after selecting the preferred type of database, the tenant need to choose between different resource configurations (CPU, RAM and storage), deployment types (either single- or multi-

deployment to have a standby instance in case of a failure or maintenance), deployment zones, database management software versions and maintenance settings. Hints and tooltips try to guide end users throughout the creation process, but due to the fact that Amazon offers a complete DBMS in a virtual machine, the end users are faced with a lot of configurable options, which can be quite confusing for the beginners. Especially for the end users who want to use only the free-tier model, it is very unclear during the navigation which of the options are eligible or which of them can easily lead to unwanted costs. After the new database instance has been created, the end users can select only actions like shutdown and restart the database server, modify and monitor the resources, but no online query interface is provided.

Azure SQL Database: Creating a new database instance is very easy and requires only a few clicks in the GUI. End users can choose whether the database shall be created on an existing server at Azure or a new server instance. Here the end users can select a server name, admin login name, password, the location and the software version of Azure SQL Database. Additionally, the end users can select the data source for the database. It can be either blank, or a sample database or a backup of another Azure SQL Database. After that, the end users select the pricing tier ranging from shared infrastructure models, the standard models with already a lot of functionalities and features, to premium packages for enterprise databases. The database can be associated to a resource group, which is a container for managing a collection of Azure resources.

The GUI generally provides a brief but complete overview of an already running database. End users can see the current workload, size and alerts. It is also possible to review the initial settings, but without changing them except for the pricing tier. The GUI extends itself to the right and enables the end users to see the information on the previous options. Everything is accessible within a few clicks as it is reasonable and the effects of actions are briefly outlined.

Amazon DynamoDB: The GUI is very clear and simple. End users can either create a new table as a starting action or choose between different helping actions like an introduction video or different “Learn More” sections, which will forward the end users to the corresponding documentation. During the creation process, guidance, hints and tooltips help the end users throughout the whole process. If a table has already been created, the end users can select a few clear actions, e.g., explore or delete the table, modify the desired throughput capacity, change the access control or create an index. The explore view is an online query interface, which also provides capabilities to edit, delete or export documents. Furthermore, comprehensive monitoring and alarm definition opportunities can be used.

Azure DocumentDB: Using the GUI, end users can select the location and the resource group. Currently, Azure DocumentDB supports only one pricing tier, which cannot be altered via the GUI. This is everything that has to be done to create a database instance; in about 10 minutes, the instance is up and running. The GUI for running database instances is similar to that of Azure SQL Database. The configurable options are as scarce as the steps to create a database instance. There are options for the consistency, roles, users and tags for manageability. But in contrast to Azure SQL Database, Azure DocumentDB offers a Query Explorer for direct querying via SQL. In regard to administrating the documents and collections of a database, the GUI lacks clear navigation in our opinion. In particular, it took us an unnecessarily long time to create a collection via the GUI and set up a database.

5. Backup and recovery capabilities

Amazon RDS: Since Amazon RDS provides each tenant as a separate DBMS in a virtual machine, Amazon offers the snapshot capabilities of their underlying infrastructure. Tenants can choose between automatic backup snapshots during the selected maintenance window or can manually create snapshots. For recovery, Amazon RDS offers a clear overview of all snapshots with the possibility to create a virtual machine instance from the snapshot, restore or delete that instance. During our experimentation with the DBaaS, we noticed another bad characteristic of the automatic snapshot mechanism that is by default active for the free-tier end users. The backups are accounted if they exceed the 20 GB free backup limit, which can happen relatively fast if a Microsoft SQL Server instance together with the default value of seven-day retention is used.

Azure SQL Database: Since DBMSs are also provided as virtual machines, a backup is based on snapshots. However, in contrast to Amazon RDS, Azure SQL Database keeps the mechanism under its own control and abstracts this functionality. The recovery capability of Azure SQL Database allows end users to select a restore point to a specific minute within a time range defined by the service tier. Copies and exports as more traditional ways of backups are also an option within Azure SQL Database. Here the end users can select or create a new Azure SQL Server when copying a database or select an Azure storage or container for exports.

Amazon DynamoDB: Compared to other DBaaS offerings, which are provided as virtual machine instances, thereby using their snapshot capabilities, Amazon DynamoDB offers no possibility to create or restore backups. However, Amazon assures that

their underlying Simple Storage Service (S3) network provides a high level of redundancy and fault tolerance so that a data loss cannot occur. Additionally, it is possible to export and import the data via a defined data-pipeline into or from an S3 bucket where it can also be downloaded [6].

Azure DocumentDB: Just like Azure DynamoDB, Azure DocumentDB does not offer any possibility to create or restore backups. It is only possible to export the data via a client application and later import this database state either via the same application or via the GUI [7].

6. Scalability, replication and availability

Amazon RDS: During the creation of a database instance, end users can choose whether they want to have a single- or multi-deployment instance. If they choose a multi-deployed instance, a standby instance will be created in another availability zone, which increases the durability and availability in case of failures. To keep the data up-to-date with the primary instance, MySQL, Oracle and PostgreSQL engines utilize synchronous physical replication, whereas Microsoft SQL Server engine uses synchronous logical replication to achieve the same result by employing Microsoft SQL Server-native mirroring technology. In addition to the multi-deployment, Amazon RDS utilizes the integrated replica functions of MySQL and PostgreSQL to create read replicas (slaves). Creating constellations, which would allow multiple write instances to improve the write throughput, is currently not available. This means that Amazon RDS offers only a scale-up solution (increasing the assigned virtual machine's resources) when it comes to higher writing demands. Furthermore, creating read replica instances requires several minutes and cannot be automated to dynamically react on higher demands [8].

Azure SQL Database: Azure SQL Database uses a partitioned database model over a large-scale, shared nothing infrastructure. Each database partition is replicated across a set of loosely coupled and dynamically assigned hosts using a custom primary-secondary replication scheme [9]. Instead of relying solely on scale-up, Azure SQL Database strives to provide scale-out and to retain the relational model and support ACID (Atomicity, Consistency, Isolation, Durability) properties of transactions within a consistency domain. To achieve this, Azure SQL Database uses an architecture of different layers, which handles clusters and the roles of each node within that cluster and replications. Azure SQL Database maintains n-safe durability by replicating each transaction across a set of replicas. At any point in time there is a single master replica and a configurable number of secondary partition replicas [9]. The use of replicas enables high availability, since a replica node can replace another node in the case of a failure. Azure SQL Database further enables geo-replicas in various locations all over the world and gives recommendations in the GUI where replicas may be best placed.

Amazon DynamoDB: As with the backup, Amazon DynamoDB abstracts things like virtual machine instances so that scaling, replication and availability options are not available to end users. Nevertheless, as the size of data grows, Amazon DynamoDB will automatically spread the data over sufficient number of virtual machines and will utilize synchronous physical replication, whereas the datacenters will guarantee high availability and durability. The only thing that can be configured is the desired throughput for a table, which is measured in read/write DTUs. To dynamically adjust the throughput to changing demands, Amazon DynamoDB provides API functions to increase or decrease the throughput with the restriction that the throughput can only be decreased twice a day but increased at any time [10].

Azure DocumentDB: Azure DocumentDB can practically scale to an unlimited amount of documents partitioned by collections, the containers for JavaScript Object Notation (JSON) documents. This means that the documents within a collection are not shared across multiple nodes but rather kept together on one database node. Collections also provide the query execution and transaction scope for all the documents contained within a collection and can get assigned a reserved amount of throughput, which is the only available configuration option regarding scaling, partitioning or availability for end users. Unlike some other document-oriented databases, which encourage the end users to store all documents in the same collection (or table), Azure DocumentDB will achieve a greater scale and performance with multiple collections when dealing with huge amounts of data [11].

Azure DocumentDB enables only a maximum of 2500 request units per second, per collection. For example, if an insert costs 20 request units, the end users would be able to do 100 inserts per second. This means that the end users can quickly run into `RequestRateTooLargeException`. Besides the exception name, the response header has a retry-after field, which tells the end users how long they have to wait before the end users can retry the operation. Even when developers know the anticipated user behavior, they have to build every database action with a reaction to this specific error and the retry treatment. Otherwise, there is a risk of developing an error-prone client application.

7. Privacy and security

Amazon RDS: The capability to create different users depends on the selected database instance. To restrict the general access to the instance, Amazon

RDS offers a lot of capabilities to assign virtual private cloud environments, security groups and security Access Control Lists (ACLs) to create particular inbound and outbound access rules by Internet Protocol (IP) address and port number. Another offered security feature of Amazon RDS is encryption, which is available for all databases but only with more expensive and faster instance classes. If this feature is selected during the creation, Amazon RDS will encrypt all corresponding data like logs and backups with a generated or customer-managed key [12].

Azure SQL Database: Azure SQL Database offers firewall rules where the tenant can configure an ACL per DBaaS. Another feature is the option to manage users and roles to enable more fine-grained access control. Furthermore, Azure SQL Database enables transparent data encryption for database instances of higher price categories. Azure SQL Database encrypts the storage of an entire database and the log files using a symmetric key called the database encryption key. That key is protected by a built-in server certificate, which is unique for each Microsoft SQL Server [13]. Additionally, Azure SQL Database enables dynamic data masking where the tenant can configure which fields may be partially obscured by the DBaaS when returned as a query result. For instance, a field containing the credit card number may be queried. In that process, the DBaaS may return a lot of X and the last four digits remain as the original ones. In most cases, the last four digits are sufficient for end users to identify their credit cards but are almost worthless for potential attackers.

Amazon DynamoDB: Amazon Web Services (AWS) Identity and Access Management (IAM) enables the tenant to grant and manage access to Amazon DynamoDB. By defining IAM policies, granting access for different users, groups and even foreign accounts from Amazon, Google and Facebook is possible. In a policy, it is also possible to define API action restrictions for end users to, e.g., enable only reading data but not editing them. Besides API restrictions, allowing certain attributes to hide sensitive information like passwords is a powerful option.

Azure DocumentDB: Azure DocumentDB enables the configuration of users and roles as well, whereas firewall settings to set up ACLs are missing. Rather, Azure DocumentDB uses keys and read-only keys to enable restricted access. Apart from that, there are no further security features available.

8. Database language richness

Amazon RDS: Amazon RDS provides all the native capabilities of the utilized DBMS for the tenants, since each tenant gets their sole database. For instance, if tenants selected a MySQL database instance, they could use all native capabilities like indices, trigger, stored procedures or access management of the underlying DBMS. To access the DBaaS, all native APIs as well as management access are possible, e.g., MySQL Workbench for MySQL or Microsoft SQL Server Management Studio if Microsoft SQL Server was deployed.

Azure SQL Database: Azure SQL Database exposes most of the functionality of Microsoft SQL Server with some restrictions. The restricted features generally fall into several classes:

- Features, which would be inherently unsafe in a shared cluster environment, such as extended stored procedures.
- Features, which may lead to excessive resource consumption within a single database that would impact other end users on the same node. (Azure SQL Database contains several resource governors to mitigate this problem but it is an area for further investment.)
- Features, which are associated with the server instance rather than a specific database. These items include server level performance and resource monitoring queries as well as access to the server error log. Azure SQL Database virtualizes many of these resources and views to provide database specific versions [7].

Amazon DynamoDB: As usual for NoSQL databases, Amazon DynamoDB provides a low-level REST API, which makes it accessible from almost every programming language or platform. Besides this API, Amazon DynamoDB provides comprehensive SDKs for Java, .NET and PHP to offer convenient and sophisticated functions to, e.g., create data out of a JSON document or to register a callback to get noticed after the action is completed. From our experience with the Java SDK, it is generally easy to use and it provides a modern and fluent syntax via method chaining.

To retrieve data, end users can either perform a scan or a query on the data. A scan conducts a full table scan and filters can be added to compare attributes with strings, numbers or binary whether they are equal to, begin with, are greater than, are less than or contain something. Performing a query takes advantage of available indices and the same compare operations of attributes can be applied. The restriction of queries is that only indexed attributes can be queried. Furthermore, both retrieving operations can get an additional parameter to specify whether an eventual or strong consistent read should be conducted.

For specifying indices, Amazon DynamoDB provides different types. A primary key has to be defined during the creation of the table. End users can choose between a hash and a combination of hash and range primary key. If the hash-range

combination is used, queries for a range attribute must also have a value for the hash attribute. In addition to the primary keys, the end users can also define secondary indices, which can be differentiated as global or local. A local secondary index can be defined as an index that has the same hash key as the table, but a different range key. A local secondary index is "local" in the sense that every partition of a local secondary index is scoped to a table partition that has the same hash key. Moreover, a global secondary index is an index with a hash and range key that can be different from those on the table. A global secondary index is considered "global" because queries on the index can span all of the data in a table, across all partitions [14].

Although transactions are very unusual in NoSQL databases, an extension library for Java Software Development Kit (SDK) offers this feature. The users can configure different isolation levels and perform multiple atomic writes within one transaction. Business logic capabilities like triggers or stored procedures are not available so that additional logic has to be implemented in applications.

Azure DocumentDB: The core of all provided APIs is the Representational State Transfer (REST) API offered by Microsoft. Besides that, other APIs for a variety of programming languages such as Java, .NET, Ruby and Node.js are available. Whereas inserts can be implemented as expected through an API call like `createDocument`, queries have to be performed as an SQL statement. Although this might be convenient for end users who are new to the DBaaS but have experience in SQL, things like querying for the database or collection to check if one exists seems a bit peculiar when otherwise an API is provided. Moreover, instead of using the collection in the `FROM` clause, a root has to be entered as a kind of wildcard and the collection has to be passed as an object reference in addition to the SQL statement to the function call.

Instead of providing traditional index functionality, Azure DocumentDB offers an automatic index feature with the choice of a lazy or consistent indexing mode. Although most NoSQL databases lack transactions and business logic, Azure DocumentDB supports both with the restriction that transactions are only possible on the server side. Azure DocumentDB supports cross-document transactions expressed as JavaScript stored procedures and triggers. Transactions are scoped to a single collection and executed with ACID semantics as all or nothing isolated from other concurrently executing code and user requests [15].

9. Pricing

Amazon RDS: End users can choose between twelve instance classes, which differ in provided number of CPU, RAM, EBS (Elastic Block Store) optimized for improved input/output and network performance, while each class has different costs for different databases. The end users can choose during the creation between different storages and deployment options, which have varying prices. Additional costs can arise for scaling and availability features like reserved instances or read replicas as well as backup storage and network traffic. In other words, the price model of Amazon RDS offers only a light abstraction of the resources and can be quite complex and costly for a demanding database architecture.

Azure SQL Database: As the number of DTUs increases, so does the power is offered by the performance level. For example, a performance level with five DTUs has five times more power than a performance level with one DTU [16]. To determine the right number of DTUs for an application, Microsoft offers a benchmark. The levels therein differ in the DTUs they provide, the database size, point in time restore and, of course, the price. For example, the basic tier with five DTUs, 2 GB database size and 7 days of point in time restore costs $0.0067 per hour, whereas the S1 tier with twenty DTUs, 250 GB database size and 14 days of point in time restore costs $0.0403 per hour. These prices are for single databases. In a preview mode, Microsoft also provides an elastic database model where the provided resources scale with the workload of the database. On an hourly basis, the needed tier to meet the workload requirements will be billed [17].

Amazon DynamoDB: In the free-tier model, end users can configure up to 25 DTUs for read and write throughputs and get 25 GB free store. One unit of read capacity allows the end users to perform one strongly consistent read or two eventually consistent reads per second for an item up to 4 KB size. Similarly, one unit of write capacity allows one write per second for items of up to 1 KB in size. If performance or storage of the free tier is not enough, Amazon offers different prices per datacenter. For instance, in the datacenter region Germany (Frankfurt) every 50 units of read capacity cost $0.00793 per hour and every 10 units of write capacity cost $0.00793. If more storage is needed, Amazon demands $0.306 for each additional GB in Germany. In conclusion, this pricing model offers a very good abstraction so that even a demanding database architecture can be calculated easily if the arising throughput is known.

Azure DocumentDB: It is billed based on the number of collections contained in a database account. Each account can have one or more databases and each database can have a virtually unlimited number of collections, although there is an initial default quota of 100 [18]. There are three price categories for collections ranging from S1 to S3. All

of them offer Solid State Drives (SSD) storage of 10 GB per collection, a 99.95% attainment of their SLAs and the already mentioned scale out limit of 100 collections. They differ in request units they can handle and, of course, in the price. For 250 request units per second in S1, the tenant has to pay $0.034 per hour. For 1000 request units per second in S2, it costs $0.067 per hour and S3 offers 2500 request units per second for $0.134 per hour. The tenant may choose to switch dynamically between the service tiers but has to remember that they are billed hourly, even if a higher tier was needed for only a few minutes [18].

10. Maintenance

Amazon RDS: Serving each tenant with a separate database instance also means that each database has to be updated separately. Amazon RDS offers an automatic minor update installation during the specified maintenance window. For major update, which could break the compatibility with interacting systems (e.g., due to API changes), Amazon RDS offers the possibility to install an automatic update manually so that creating a new instance with the newer major version and migrating the data to it are not necessary. All updates will result in downtime of the database if no multi-deployment is enabled. Regardless of whether an update requires an instance restart, Amazon RDS will restart the instance even though it can cause a potential additional downtime.

Azure SQL Database: Its server cluster system is aware of the layout and topology of all nodes, partitions, racks, and physical network routes. This allows the system to coordinate software upgrades in a way that allows for high data availability during the upgrade [9]. This means that Azure SQL Database handles updates the same way as Amazon RDS in a multi-deployment mode. It updates every replica one after the other and another replica will replace the currently updated one. This results in very high availability of the stored data.

Amazon DynamoDB: Downtime or other consequences due to maintenance are nearly non-existent with Amazon DynamoDB. In fact, end users do not have to worry about maintenance because it is applied in secret. With its peer-to-peer architecture and the replication of the data across multiple nodes, Amazon DynamoDB can reach an availability and request success rate of 99.9995% over a twelve-month period [19].

Azure DocumentDB: Unfortunately, we did not find any information about how Microsoft manages updates of the DBaaS. But it is very likely that Azure DocumentDB also uses replicas and a similar architecture as Azure SQL Database, which would enable a seamless update just like in Azure SQL Database.

11. Performance

Performance is typically one of the top evaluation criteria for DBMSs. Since DBaaS is still relatively new, many customers expect that DBaaS providers will continue over time to improve functionality, usability, scalability, availability and security of their offerings. Many customers are willing to use DBaaS offerings that may not entirely meet an end-user's wish-list specifications in those areas. However, they typically do want to get the best performance possible. In other words, an end-user's level of satisfaction with a DBaaS offering is largely determined by its performance. Since performance is so important, we decided to include it into the list of evaluation criteria, thereby extending our previous work [20].

Evaluating performance involved:

- populating a database (one for each DBaaS offering) with test data;
- developing a test application;
- running the test application against the test data;
- measuring the test application's performance.

11.1. Test data

Each of the four databases stores information on customers, including the customer first names, last names, usernames, passwords, phone numbers and addresses. Furthermore, the databases keep a list of items ordered by customers during their visit to a web shop, including the item names, descriptions and prices. The databases do not keep a track of the amounts of individuals of an item, so that an item can be part of more than one order or none.

A customer can have zero or more orders. In its turn, each order belongs to exactly one customer. Since a customer may have never placed an order, an address belongs to exactly one customer but to any number of orders. An order without any item would be useless. That is why an order needs to have at least one item, although the item can be associated to an arbitrary number of orders. This constellation implies that the database can hold a lot of items without a corresponding order.

An address is described by street, house number, zip code, city and country. As mentioned above, addresses are used by both customers and orders. A customer has at least one address, whereas an order has exactly two addresses: one for the delivery of the items and another for the billing. On the one hand, this leads to redundant storing of addresses, but if the address of a customer or some attributes of an item were changed, these changes would not affect the order so that the original state of that order could always be retrieved.

In addition to the delivery and billing addresses, orders have the time of their placement by customers.

11.2. Test application

The test application provides a GUI for choosing one of the four databases (see Figure 1).

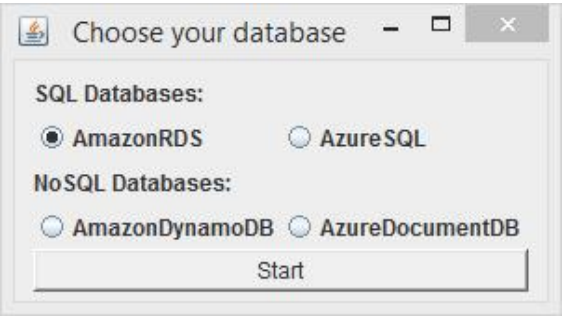

Figure 1. Selection of DBaaS offerings

After that, the application retrieves all information on customers, orders, items and addresses from the selected database. The GUI also offers the functionality to add, delete and update this information (see Figure 2).

For adding a dataset, the application will detect the currently selected tab (which corresponds either to customers, orders, items or to addresses) and show an appropriate form where we can enter new information and press the `Insert` button. Updating a dataset can be done by clicking on a cell, editing the data and pressing the `Update` button. Deleting a dataset is just as simple as follows: we can select a row and remove the dataset by clicking on the `Delete` button. The flush function deletes all data from the database, not just the selected datasets. This function is also used by the import function, which at first flushes and then inserts all data.

Figure 2. Displaying information from selected database

11.3. Test results

We used two different datasets to evaluate performance of the selected DBaaS offerings. The first dataset was a smaller one with only 250 customers, 200 orders, 100 items and 380 addresses. The second dataset was 30 times larger than the first one. In particular, it contained 7500 customers, 6000 orders, 3000 items and 11400 addresses. We expected that the times to process the second dataset would also increase 30 times.

Figures 3 and 4 summarize the results of our performance tests.

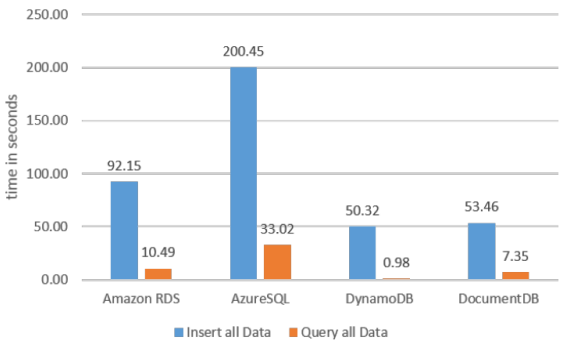

Figure 3. Performance comparison with first (small) dataset

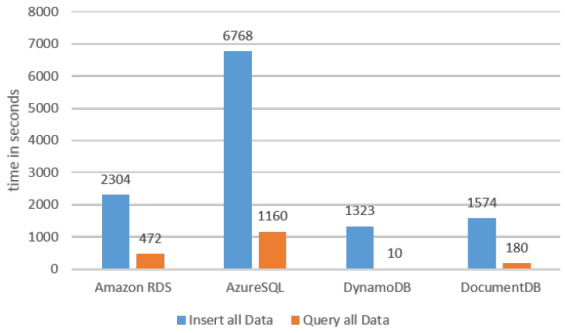

Figure 4. Performance comparison with second (large) dataset

Amazon RDS: In Amazon RDS, the achieved performance depends on the selected virtual machine configuration. In the free-tier model, we could only use the micro-instance with 1 vCPU, 1 GB RAM and low network performance. However, the overall performance was significantly better than that of Azure SQL Database. The required time for the second dataset prolonged to 25 times of the smaller dataset for the import. This was under the predicted 30 times increase. However, the required time for querying was increased 45 times. All the offered monitor features did not show any noticeable changes in the latency and read throughput during the querying. Although Amazon RDS showed the best performance among the relational databases, our comparison of the test results with the non-relational databases revealed that querying different tables and joins caused a big overhead.

Azure SQL Database: The measured times with Azure SQL Database were drastically slower, even

though we used a 25 DTU performance level. Additionally, looking on the online monitoring features of Azure showed that the application utilized 5-10% during writes and 45-85% during reads of the available DTUs. Although this explains the high benchmark times, we could not find any justification why the resource utilization was so low. Moreover, the rising processing times between the first and the second datasets were longer 33 times for the import and 35 times for the querying though those numbers were still within an acceptable range.

Amazon DynamoDB: As per free-tier model, we configured 25 DTUs for read and write throughputs. During our performance tests with the second dataset, we noticed that Amazon DynamoDB reduced the throughput down to one DTU as we exceeded the allowed requests over a five-minute window. However, we could catch such information in form of an exception and re-adjust the number of DTUs as desired. Despite this short downgrade, Amazon DynamoDB achieved the best results in all categories. Looking at the differences between the performance for the first and the second datasets revealed that while the time for the import increased 26 times, the querying required only 10-fold of the times of the first dataset.

Azure DocumentDB: As with Azure SQL Database, we chose 25 DTUs for our performance tests. Although the insert time with the first dataset was only slightly higher than that in Amazon DynamoDB, especially the test with the second dataset showed enormous differences between the two. This was due to the service tiers' limitations of request units per second. With the tier S1 of data collection, the limit was 250 request units per second. The average cost of inserting customers, orders and items was 13, 30 and 6 request units, respectively. With that in mind, we calculated that we could insert 19 customers, 8 orders or 41 items per second. Those numbers amounted to nearly the desired result.

12. Conclusion

In this paper, we evaluated four DBaaS offerings. A GUI was what we came in contact with first. While the GUI of Amazon RDS is rather convoluted, the GUI of Azure SQL Database is clear and comprehensible. As to the GUIs of the NoSQL databases (i.e., Amazon DynamoDB and Azure DocumentDB), they are similarly structured and offer a query interface, which is missing in the relational databases. All the GUIs provide options to configure the services and monitor them.

Next, we analyzed the backup and restore functionalities of the selected DBaaS offerings. The relational databases are based on separate database architectures and offer snapshot capabilities to backup and restore the databases to a state in any given point in time. The NoSQL databases automate and abstract this functionality. The only way to backup and restore interactively is to export the data and import them later.

However, probably the most interesting criterion was the implementation of replication, scaling and measures to provide high availability. Amazon RDS uses the built-in replication mechanisms of the utilized DBMS to set up a distributed database with read replicas though a tenant has to create read replicas via a one-click action using the GUI. However, this service does not offer write replicas. On the contrary, Azure SQL Database abstracts the replication per datacenter and automatically scales within the boundaries of the selected service tier. The tenant can select a geo-replication to another datacenter using the GUI to lower latency in other regions of the world and thus, enable higher availability. Amazon DynamoDB abstracts replication, scaling and availability options. It guarantees to provide a sufficient number of virtual machines, synchronous replication and high availability. Azure DocumentDB can practically scale to an unlimited number of documents partitioned by collections. Currently, the limitations are 10 GB per collection and 100 collections per database. In our experiments, we encountered availability problems due to the limited number of requests that Azure DocumentDB could handle. Although we were able to resolve this matter, the official solution to this limitation produced a bloated and error-prone code.

Despite their obvious advantages over DBMSs, security concerns still hinder the success of DBaaS the most. That is why we looked into features that would provide a safe environment for the tenant's data. Amazon RDS depends on its cloud security measures to access to the service. The database itself, however, has to be protected by means of the utilized DBMS. For all offered DBMSs, Amazon also provides encryption for the stored data but only within more expensive instance models. Azure SQL Database can be secured via configurable ACLs as well as via a preset of roles with varying capabilities where users can be associated to one or more roles. Additionally, sensible data can be obfuscated on querying to reduce deductions for a potential attacker from query results. Furthermore, Azure SQL Database offers transparent data encryption for database instances of higher price categories. In contrast to Amazon RDS, Amazon DynamoDB takes full advantage of the cloud security measures of Amazon. It allows tenants to create users and groups, even from Google and Facebook, to restrict access and API functionalities through policies. But it is worth noticing that encryption is not available yet. Regarding security features, Azure DocumentDB lacks in various aspects. The only feature available is

the access restriction via read-only keys. There are no groups, roles, ACLs or encryption.

To replace DBMSs and to present a viable option, DBaaS offerings have to provide the same database functionality. Therefore, we also analyzed the capabilities of all competitors in that regard. While Amazon RDS provides all native functionalities of the utilized DBMS, Azure SQL Database abstracts the database and therefore allows only a subset of the SQL capabilities due to restrictions that have to be kept in shared environments. The NoSQL databases provide a REST API and additional SDKs for major programming languages, though they differ in their query capabilities. Whereas Amazon DynamoDB uses the API to query or scan for documents, Azure DocumentDB uses SQL for its queries. Additionally, Amazon DocumentDB creates indices automatically, whereas Azure DynamoDB enables to set them manually as required by application.

As already mentioned, Amazon RDS offers only a light abstraction of the resources, which reflects in the pricing. It gets as complex as the architecture that it represents. All the pricing of the other services is based on a blended measurement of hardware resources or a mix of properties of the DBaaS. The providers try to abstract the hardware requirements to meet the demanded database performance. This facilitates an easier calculation of costs for tenants.

In the long run, it was also interesting to know about the maintenance features of the DBaaS offerings. Whereas Amazon RDS requires that the tenant should set a maintenance window in which the instance will be cut off to install updates and take backups, Azure SQL Database abstracts the maintenance and handles it secretly without the tenant taking notice of this. Both Amazon DynamoDB and Azure DocumentDB follow the abstraction of Azure SQL Database and therefore, also enable seamless updates and backups.

Finally, in our performance evaluation, the fastest DBaaS offering was Amazon DynamoDB, followed by Azure DocumentDB and Amazon RDS. The slowest DBaaS offering was Azure SQL Database. Even though we used the free-tier model for Amazon RDS with very low performance, Azure SQL Database did not reach the expected results because the test application could not fully utilize the provided resources.

13. References

[1] V. Mateljan, D. Cisic and D. Ogrizovic. “Cloud database-as-a-service (DBaaS).” In Proceedings of the 33rd International MIPRO Convention, 2010.

[2] Amazon RDS, http://aws.amazon.com/rds/.

[3] Amazon Dynamo DB. http://aws.amazon.com/de/dynamodb/.

[4]Azure SQL Database, http://azure.microsoft.com/services/sql-database/

[5] Azure DocumentDB, http://azure.microsoft.com/services/documentdb/

[6] “Cross-region export and import of DynamoDB tables.”https://aws.amazon.com/de/blogs/aws/cross-region-import-and-export-of-dynamodb-tables/

[7] “Import data to DocumentDB”, https://azure.microsoft.com/de-de/documentation/articles/documentdb-import-data/

[8] “Amazon RDS multi-az deployments”, http://aws.amazon.com/rds/details/multi-az

[9] D. G. Campbell, G. Kakivaya and N. Ellis. “Extreme scale with full SQL language support in Microsoft Azure SQL Database.” In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD, pp. 1021-1024, New York, NY, USA, 2010. ACM

[10] Amazon DynamoDB FAQs, http://aws.amazon.com/dynamodb/faqs/

[11] A. Liu, “Scaling a multi-tenant application with Azure DocumentDB.” http://blogs.msdn.com/b/documentdb/archive/2014/12/03/scaling-a-multi-tenant-application-with-azure-documentdb.aspx

[12] “Encrypting Amazon RDS resources”, http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Overview.Encryption.html

[13] “Transparent data encryption with Azure SQL Database.” https://msdn.microsoft.com/library/dn948096/

[14] “Improving data access with secondary indexes in DynamoDB”, http://docs.aws.amazon.com/amazon-dynamodb/latest/developerguide/SecondaryIndexes.html

[15] “DocumentDB server-side logic and transactions.” https://azure.microsoft.com/en-gb/documentation/articles/documentdb-faq/

[16] “Azure SQL database service tiers and performance levels.” https://msdn.microsoft.com/library/azure/dn741336.aspx

[17] “Azure SQL Database pricing”, http://azure.microsoft.com/en-us/pricing/details/sql-database/

[18] “Document DB pricing”, http://azure.microsoft.com/en-us/pricing/details/documentdb/

[19] S. Sivasubramanian, “Amazon DynamoDB: a seamlessly scalable non-relational database service.” In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 729-730. ACM, 2012

[20] I. Astrova, A. Koschel, C. Eickemeyer, J. Kersten, N. Offel. ”DBaaS comparison: Amazon vs. Microsoft”, in

Proceedings of the International Conference on Information Society (i-Society), pp. 14-19, IEEE: UK, 2017.

14. Acknowledgements

Irina Astrova's work was supported by the Estonian Ministry of Education and Research institutional research grant IUT33-13.

