IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 13, 2018, accepted January 8, 2019, date of publication January 18, 2019, date of current version February 12, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2893333

Performance of OnPrem Versus Azure SQL Server: A Case Study

ROBERT GYŐRÖDi[1], MARIUS IULIAN PAVEL[2], CORNELIA GYŐRÖDi[1], AND DOINA ZMARANDA[1]

[1]Department of Computer Science and Information Technology, University of Oradea, 410087 Oradea, Romania
[2]Department of Computer Science and Information Technology, Faculty of Electrical Engineering and Information Technology, University of Oradea, 410087 Oradea, Romania

Corresponding author: Robert Győrödi (rgyorodi@uoradea.ro)

ABSTRACT This paper presents a comparative study between on-premises databases and cloud databases regarding the response time of the database. It focuses on presenting the advantages of storing data and applications in the cloud and, of course, on managing it in comparison with managing the same data and applications locally on one or more physical machines. A Microsoft Azure account was created to manage the database that is stored in the cloud. To achieve comparative results, a specific testing architecture that uses a Universal Windows Platform app was created and used. The application is running locally on multiple physical machines and works with that database to extract data, operate, and upload new data. For local tests, the database was stored locally on a server, with and without replication, and for cloud tests, it was stored into a server in Central US. This paper provides a practical approach that could be used for examining the performance of basic database operations when dealing with a different number of user situations. As a result of tests carried out, we will highlight the many advantages of cloud data storage, such as data accessibility, speed, security, automation, and disaster recovery, and we will also try to offer an answer to the most common and important question: ''Why cloud?''

INDEX TERMS Cloud, database, data storage, Microsoft Azure, Microsoft SQL Server, on-premises.

I. INTRODUCTION

Developing technologies and the growth of the amount of information people want to have access quickly from anywhere at any time has led to the creation of cloud technology. Whether we are talking about pictures from the smartphone, a collection of .pdf, .doc files, software or maybe favorite movies, the data requirements keep increasing day by day. According to International Data Corporation, it is predicted that big data is growing at an annual rate of 60% for both structured and unstructured data [1].

In this context of increasing the complexity of user requirements, cloud represents an alternative for storing, enabling access to files from everywhere and from any device that is connected to the internet and capable to process or display those files. Consequently, the cloud is a good solution that can assure safety of data, being almost always at hand [7].

Cloud computing is an Internet-based computing model which has gained significant popularity in the past several years as it provides on demand network access to a shared pool of configurable and often virtualized computing resources typically billed on a pay-as-you-use basis [6].

The term cloud computing is somewhat difficult to define precisely. A traditional definition is that it provides, at an actual monetary cost to the end-user, computation, software, data access and storage that requires no end-user knowledge about physical location and system configuration [5].

Currently, there are several used and well-known cloud providers available on the market, such as Amazon Web Services, Microsoft Azure, EnterpriseDB, Google Cloud Platform, MongoLab, RackSpace [1].

Amazon Web Services [9] has a variety of cloud-based database services, including MySQL, Oracle or SQL Server as relational database, and Amazon DynamoDB, the Amazon solid-state drive, on the NoSQL side [8]. The direct competitor of Amazon is Microsoft with its Microsoft Azure [10] that uses SQL Server technology to provide a relational database, allowing customers to either access an SQL database on its cloud, or hosted SQL server instances on virtual machines. Microsoft also emphasizes on hybrid databases that combine

data both on a customer's premise and with the Azure cloud through SQL Data Sync. Microsoft has a cloud-hosted NoSQL database service named Tables as well, while Blobs (binary large object storage), are optimized for media files such as audio and video.

Another cloud database is EnterpriseDB which is focused on the open source PostgreSQL database and provides the ability to work with Oracle database application. Garantia Data offers a gateway service for users to run open source Redis and Memcached in-memory NoSQL databases services in Amazon Web Services public cloud. Also, Google offers Google Cloud SQL that is centered on two major products: Google Cloud SQL, which Google describes as a MySQL-like fully relational database infrastructure, and Google BigQuery, an analysis tool for running queries on large data sets stored in its cloud [1]. MongoLab is another cloud provider in the NoSQL world where there are a variety of database platforms to choose from, including MongoDB; also, Rackspace represents an interesting alternative, with its virtualization of Cloud Databases that allows higher performance of the database service compared to the case it was run entirely on virtualized infrastructure [1].

Regardless of the chosen option, the alternative of cloud storage presents significant advantages in terms of availability and safety of data, but in terms of performance related to data access several issues have to be considered.

When comparing traditional database management systems with cloud databases, existing studies were merely focused on comparative analysis from the architectural point of view, starting from the basis that cloud technology implies virtualization and emphasizing differences in the internal model of the two approaches [13]. For example, the overhead induced by running a database over a virtualized environment was approached in [15] where possible solutions were presented. Scalability issues in cloud computing database architecture are analyzed in [17]. Also, challenges regarding the use of databases as a service in cloud computing were investigated in [11], [12], and [16]. Other studies investigate how relational databases are performing in the cloud environment, focusing on the existing challenges when making the system highly available and scalable and questioning whether running databases in the cloud really provide operational advantages [14].

In this idea, the present study aims to conduct an experimental investigation based on a quantitative approach for comparing the performance of a cloud database to a traditional database. Response time is considered the metric to realize the comparison. A consistent methodology is followed during the study: the relational database is designed, normalized, optimized and deployed both into a cloud and to a traditional environment; furthermore, a specific testing architecture was developed for running the tests and comparing database performance results depending on different load contexts both when it is used locally and in the cloud.

The paper is organized as follows: chapter II describes the architecture developed for testing: the UWP application, the specific API and database structure. In chapter III several performance tests were run, on-premises and on different cloud pricing tiers, and the obtained results were analyzed. Furthermore, chapter IV makes a price comparison between on-prem and cloud approaches. Finally, chapter V resumes the conclusions of the study.

II. TESTING ARCHITECTURE

The testing architecture used in the present study is presented in Fig. 1. For the cloud approach, the selection process considers several options from existing alternatives, but finally Microsoft Azure technology was chosen because it allows using the online portal for the management of all the services that we want to use [18].

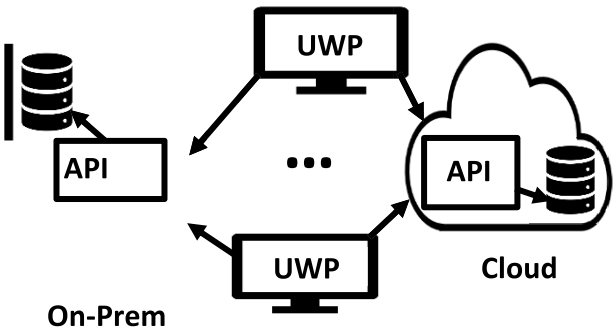

FIGURE 1. Testing architecture.

The proposed testing architecture implies also the development of a Universal Windows Platform (UWP) testing application capable to work with the database through an API, both in the cloud and on-premises. This application represents the main component of the testing architecture; a specific module of the application is used to evaluate the response time of each basic database operation.

For local approach, the specification for the physical machine for running the local database on is presented in Table 1.

TABLE 1. Configuration of physical machine.

OS	Windows Server 2012 R2 x64
Processor	Intel(R) Core(TM) i7-7700HQ CPU @ 2.8 GHz Quad Core (8 threads)
RAM	32.0 GB (2400 MHz)
Hard Disk	SSD 970 EVO 1TB

A physical server is used for storing the on-premises database and two other physical servers for hosting the web API. To obtain a consistent comparison, the on-premises architecture uses identical configuration as the cloud architecture, except that, in that case the entire equipment is running on-premises. Also, to locally obtain full SQL Server fault tolerance, the on-premises database is replicated to

another server, with same configuration. Thus, we covered the disaster recovery problem (fault tolerance) that the cloud architecture already did [21].

Microsoft standard replication mechanism was used for configuring replication between the two fully connected servers [20].

A. UWP APPLICATION DEVELOPMENT

UWP came with Windows 10 as part of the core, and it provides a common app platform available on every device that runs this version of Windows [2]. When creating a Universal Windows Platform application, a single app package that it is capable to be installed onto many devices will be created (mobile phones, computers, Xboxes, HoloLens or other devices with IoT such as physical devices that are embedded with electronics, sensors, software and network connectivity that permits these devices to collect data from the environment).

To obtain the fairest comparison, the UWP app was created to be capable to work with data both locally and inside cloud, using an Azure account. The app name is Restaurant and as the name suggests it allows running an actual restaurant, allowing the placement of new orders by the users, managing products or categories of products, managing tables of the venue, orders and even users. All this information is recorded in the database. By using large quantities of data, the app could be more or less responsive; therefore, database access performance represents an issue.

The entire application is written in C# using Visual Studio and is functioning on any kind of device that is running a Windows 10 operating system. It consists of three modules, two of them being separate projects and the third being the database, each of these with special capabilities. The first module, called Restaurant, is designed for viewing the tables of the venue, managing them, placing new orders and editing them.

The second module is called Orders, and it is used by users from the kitchen or the bar. It allows users to view the placed orders or edit an order's status. Also, here users can search for orders by order number, table number or the name of the one who placed the order.

The third main module of the application is called Settings, and it is designed to be used by the administrator of the application to add new products, associate a product to a product group, delete a product or a product group or modify any of these. Also, there is the possibility to add a new user and delete or even edit an existing one.

The Restaurant app is capable to work with the database through an API, both in the cloud and on-premises, thus allowing making several tests to view the differences between these two approaches. In the Settings module of the app, a page that allows us to measure the database procedure's execution time was created. This module allows also changing the number of users that are using the database at the same time, for analysis purposes. All the results were saved locally in a .txt file named Times.txt and could be used for further performance evaluation assessment.

B. API DEVELOPMENT

The second part of the architecture is composed of an API that exposes a set of functions and methods. API [3] is a web application that is called by the Restaurant application using HTTP calls; furthermore, API will call the database using stored procedure calls.

The API project is structured in controllers and every controller has an associated data context within which is all the logic to work with the database, receive the answer from it, and convert the data in objects and sending them back to the Restaurant application to use them.

To observe the differences between cloud and local storage, the API is published both locally and in Azure Cloud. Since the API project is just a simple web project, publishing it locally means that the Restaurant UWP app will call the methods inside it using localhost. Therefore, the Restaurant app will need a connection string, named ApiURL, which is composed from the API public address and the port that the application is listening for HTTP calls. It is initialized before the app start and in case that API is published locally it will have the following structure:

```
string ApiURL = "http://pcname:4962";
```

Here the http://pcname means that the API is published locally, on a server inside an intranet network, where the API is hosted by Internet Information Services. The number 4962 is the port number that the API is listening for HTTP calls from Restaurant application that is set inside Internet Information Services when the API is published. The default port for HTTP calls is 80 and it is required because of TCP protocol.

When publishing API locally a new web site is required and therefore IIS Manager from Windows was used for creating this.

Publishing API inside an Azure account is different from the local one because no port is needed, there is only the site address. Consequently, the corresponding connection string is like the following:

```
string ApiURL = "http://apirestaurant.azurewebsites.net/
```

Login into an Azure account could be done by using an e-mail address and a password. Afterwards, a simple web application that will be the basis for the Restaurant API project should be created. After web app creation in Azure, the publish profile which is created from a file that contains all the credentials to be used when publishing the Restaurant API in cloud should be downloaded. Once the web app in cloud was created, then the Restaurant API could be published using Visual Studio. Publishing in the cloud is like publishing application locally.

After publishing the app this one is stored in the cloud and it can be used in the same way as the one that is published locally. It is very important that the connection to the database in the API app that is now in cloud to be set for the database from cloud; this, in our case, it was very simple

because before publishing the app in cloud we just changed the connection string with the necessary connection to the cloud database.

C. DATABASE DEVELOPMENT

The third part of the architecture is represented by the database which is composed of tables, functions and stored procedures. All data bits inside it are connected using primary keys and there are tables for every object type from the app (e.g. tables, orders, products, extra-options and more).

The database project was created using Microsoft SQL Server Management Studio 2017, where from the *Databases*section, we created a blank database. As the *Restaurant*app functionalities increased, the database was populated with new tables for storing data and new stored procedures that were called from the API to answer the requests from the Restaurant app.

As it was done with API publication, storing the database locally and then in the cloud was needed to be done. For local storage, a static server that physically is situated at a small distance from our app was used. The connection to it was assured over internet inside a local network. The server is running Windows server 2012 R2 on 64-bit and inside it there is an instance of Microsoft SQL Server 2017 that is capable to accept connections over the internet.

From the beginning of this project the database was developed on this server by adding tables and stored procedures as needed. After the creation of database was finished, using the Management Studio tool, the schema of the database was extracted in order to further upload it to the Azure account.

Two identical servers with identic specifications (as presented in Table 1) were used to ensure replication on the local architecture. Transactional replication mechanism between servers was implemented, as described in [20]. Consequently, a publisher server was used; the server is called by our API in order to execute the target operations. At every transaction, before closing it, a message to a subscriber server will be sent in order to sync data between those two servers. We decided to replicate everything on our database, tables and stored procedures, similar to the mechanism used in the cloud architecture.

For storing into the cloud, a new server where the database will be stored was also created together with a username and a password to serve for login. Afterwards, the schema of the locally database with the same name *Restaurant* was added in Azure account to be accessed by the app. Using the Azure portal, a new SQL Database was created and added into the Azure Server by using the prior loaded database schema.

After creating the database in the cloud, the credentials that were just set could be used to login into database and use this to login on the Azure server from local Microsoft SQL Server Management Studio just like it could be done locally.

D. SCRIPTS AND CONFIGURATIONS

For comparing cloud storage with local storage performance, the following basic database operations were implemented into the *Restaurant* app: SELECT data from database; INSERT new data inside database; UPDATE data inside database; DELETE data from database.

All these operations were monitored to obtain real numbers and to observe the advantages and disadvantages between these two approaches.

For example, for adding products inside database (INSERT), a functionality that it is capable to send to the API HTTP requests that will result in adding data inside database, was added to the app.

```
List<Task> tasks = new List<Task>();
for (int i = 0; i < numberOfTasks; i++)
{
   tasks.Add(Task.Run(() =>
        AsyncOperationXX(Task.CurrentId)));
}
await Task.WhenAll(tasks);
```

To simulate multiple user's parallel calls to database, a list of tasks was created and run concurrently, asynchronous in the *for* loop. At every *for* loop iteration, the specific called method *AsyncOperationLocally()* or *AsyncOperationCloud()* creates a thread that will call the API and it will be used to call a stored procedure inside database in order to get the execution time of the operation. Inside database we already have a table named *Result*, where after each operation inside database, the time we get will be saved in that table. By making an average between all results stored here, we obtain the average execution time.

Stored procedures were implemented inside the database for every call from application. For measuring the execution time, *sys.dm_exec_procedure_stats* system view was used to provide information for the last operation made on database or for a specific stored procedure.

Moreover, this system view provides aggregate performance statistics for cached stored procedure. Consequently, to obtain accurate results, the implemented stored procedures were run multiple times. From this system view multiple columns were used, but the most important column is *d.last_execution_time*, which represents the elapsed time, in microseconds, for the most recently completed execution of the named stored procedure. The implemented query looks like bellow:

```
INSERT INTO dbo.Result (ThreadID, PCName,
NumberOfThreads, Opperation, ExecutionType,
ElapsedTime, ExecutionCount, AverageExecutionTime,
LastElapsedTime, OperationTime)
SELECT @p_taskID, @p_pcName, @p_tasksNumber, 'DELETE',
@p_executionType, d.total_elapsed_time,
d.execution_count,
d.total_elapsed_time/d.execution_count,
d.last_elapsed_time, d.last_execution_time
FROM sys.dm_exec_procedure_stats AS d
where d.database_id = @databaseID AND
OBJECT_NAME(object_id,
database_id)='StoredProcedureName' AND
d.last_execution_time > '2018-11-19 00:00:00'
```

To assure, as much as possible, a consistent comparison, it is important to mention that the same database, with same

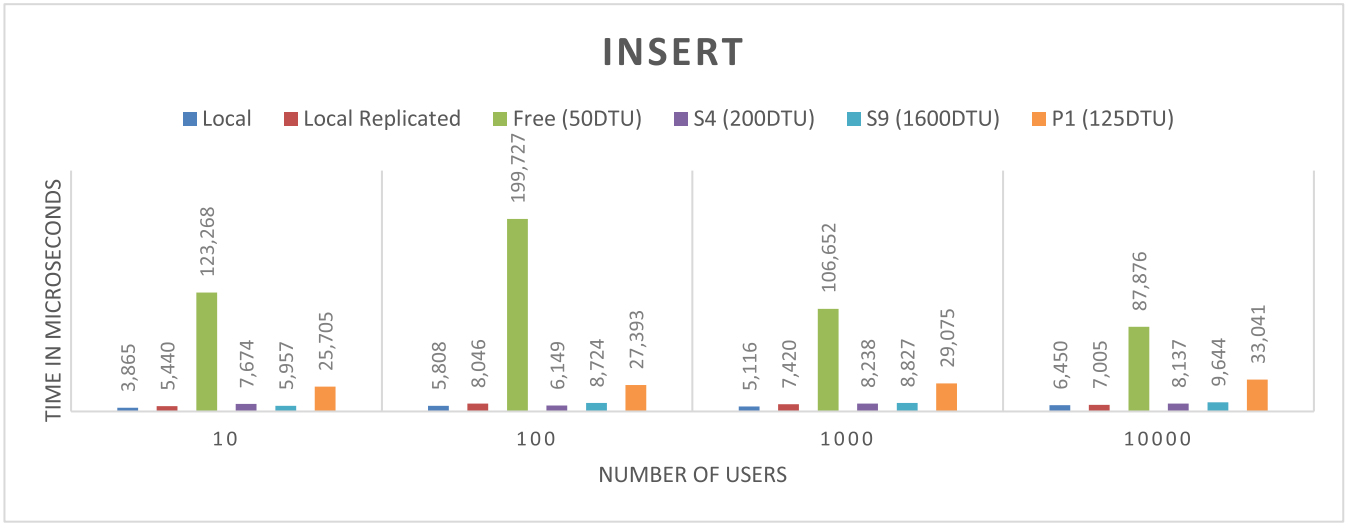

FIGURE 2. INSERT statement experimental results.

tables, same stored procedures and no data inside the Product table were used both locally and inside Azure.

The API is the same for both cases, the only difference is that one is published locally inside Internet Information Service, and the other one is published in cloud, inside the Azure subscription.

For the cloud part, multiple pricing tiers were used, because we want to highlight the time responses for each operation in comparison with on-premises architecture. On this side we will talk about DTUs. The amount of resources is calculated as a number of Database Transaction Units (DTUs) that it is a bundled measure of compute, storage, and IO resources for databases from cloud.

At the beginning we started with the Free Pricing tier, were we had an SQL server with 5DTUs (Database Transaction Units [19]) and one single database and up to 32Mb. Because of the small capacity of database, we had to backup, delete and recreate the database frequently. After this, we used Standard 4 (S4) pricing tier with 200 DTUs, and afterwards Standard 9 (S9) pricing tier with 1600DTUs, and in the end we used Premium tier (P1) with 125 DTUs. Certainly, by increasing database pricing tier, the costs were increasing because there are many resources that were used [4].

III. PERFORMANCE TESTS

Queries that include basic SQL operations were used to compare local and cloud approaches during the performance tests: INSERT, SELECT, UPDATE and DELETE. For each operation a specific query for each stored procedure was run and execution time was calculated for each operation type accordingly. The approach implies calling multiple times each query with a different number of users; consequently, a huge number of calls were made for each query and at the end an average to be as accurate as possible within the study was calculated.

The script for INSERT statement used to observe the execution times inside database is the following:

```
DECLARE @details NVARCHAR(500) = 'Lorem Ipsum Dolor…'
INSERT INTO Produse.Product ([Name], Price, Details,
HealthyState, ProductGroupID, TVAID, Capacity,
MeasureUnitID, [Status], Available, ProductTypeCode)
VALUES('Product 1', 12.0, @details, 1, 2, 3, '250', 1,
1, 1, 'PROD'), ('Product 2', 10.0, @details, 1, 2, 3,
'200', 1, 1, 0, 'PROD'), ('Product 3', 8.0, @details,
1, 5, 3, '150', 1, 1, 1, 'PROD'), ('Product 4', 6.0,
@details, 1, 3, 3, '100', 1, 1, 0, 'PROD'), ('Product
5', 4.0, @details, 1, 4, 3, '50', 1, 1, 1, 'PROD')
```

The execution time was obtained by running the following query:

```
DECLARE @databaseID INT = (SELECT
DB_ID(N'AZURE_LOCAL'));
SELECT d.last_execution_time AS 'ExecutionTime'
FROM sys.dm_exec_procedure_stats AS d
WHERE d.database_id = @databaseID AND
OBJECT_NAME(object_id, database_id) = 'usp_*Products'
```

where *usp_*Products* represent the name of the stored procedure that we want to observe, and it will be replaced by the following data, based on the operation that we will execute:

- *usp_InsertProducts*, for INSERT statement
- *usp_SelectProducts*, for SELECT statement
- *usp_UpdateProducts*, for UPDATE statement
- *usp_DeleteProducts*, for DELETE statement

During each query, the number of database users was modified, starting with ten users, then to one hundred, one thousand, and finally ten thousand users, for analyzing the performance for different load situation.

The results obtained with **INSERT** statement for all six testing environments that we used are presented in Figure 2.

As it is shown in Figure 2, there are few significant differences between Azure pricing tiers and on-premises architecture.

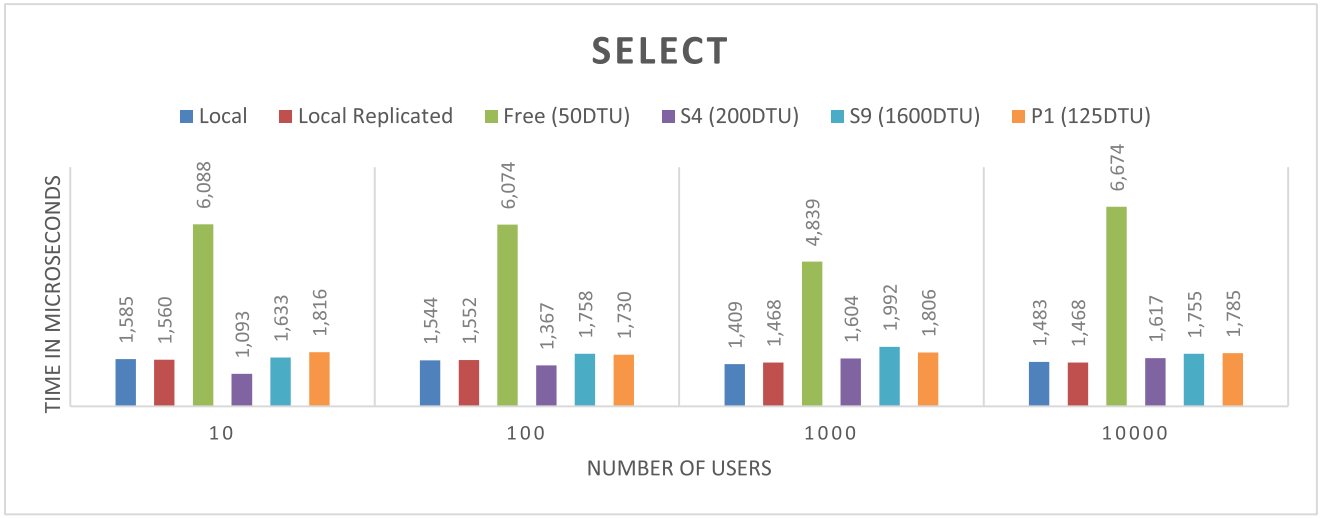

FIGURE 3. Select statement experimental results.

The smallest value is the best response time average, and in our case, we can easily observe that the Azure Free pricing tier is the slowest one, in all cases, with all number of users. Next one is Azure P1 with fair results, somewhere around 28,000 microseconds. Azure S4 obtained the best results when we are looking to overall cloud architectures, with an average time of 7,549 microseconds and close to it was Azure S9 with an average time of 8,288 microseconds. The fastest options here, seems to be the on-premises architecture, with a continuing proportional increase to the number of users. Using local replication results in slightly increased times when comparing to the non-replicated architecture, i.e. 6,977 versus 5,309 microseconds.

We can easily see that the Azure architecture is following the same proportional growth as the on-premises architecture, but the only difference here is that the Cloud architecture is slower than local one, but local architecture is getting closer to cloud one, as we increase the number of users.

Consequently, we can say the on premises architecture is faster than some Azure pricing tiers with only a few microseconds. Even with that we can say that cloud architecture is a little bit more constant than the local one.

The procedure used for SELECT statement implies selection of the first 1000 active products, by making an INNER JOIN with *ProductGroup* table and a LEFT JOIN with *VAT* table:

```
SELECT TOP 1000 p.ProductID, p.[Name], p.Price,
p.Capacity, p.HealthyState, pg.Description, t.[Value]
FROM Produse.Product p
INNER JOIN Produse.ProductGroup pg ON
pg.ProductGroupID = p.ProductGroupID
LEFT JOIN Preturi.TVA t ON t.TVAID = p.TVAID
WHERE p.[Status] = 1
```

The results obtained with **SELECT** statement are presented in Figure 3. It can be easily observed that the results are a little bit different than for the **INSERT** statement.

The Azure Free pricing tier is again the slowest option of all others, but it is followed by on-premises architecture with an almost constant result, somewhere around 1,508 microseconds as an average for all scenarios (replicated and not replicated).

The other three Azure pricing tiers seems to have almost similar results except Azure S4 for 10 and 100 users scenario, where it seems to be better than on-premises architecture with around 0.4 microseconds. We can easily see that the results for SELECT statement are pretty constant, the local environment being only a little faster
(around 0.4 microseconds) than cloud, when we refer to a bigger number of users, like 1000 or 10000.

The procedure code used for **UPDATE** operation is based on a simple update statement where the product's details are updated with new data:

```
UPDATE TOP(100) p
SET p.Price = 44.4
FROM Produse.Product p
INNER JOIN Preturi.TVA t ON t.TVAID = p.TVAID
WHERE (p.Capacity = 100 OR p.capacity=50)
      AND t.TVAID = 3
```

For the **UPDATE** statement, we have to mention that this operation is a heavy CPU operation. Therefore, as it is shown in Figure 4, the best results for cloud architecture were obtained by using the Azure S9.

Also, in this case Azure Free pricing tier is again the slowest one, with an average of 1.5 seconds. Because of the small number of DTUs, Azure Premium 1 tier is placed on the fifth place, but at a big distance of Free pricing tier, and closer to other tiers.

Because of the biggest number of DTUs, Azure S9 obtained an average of 31,782 microseconds comparing to S4 tier which obtained an average of 38,964 microseconds. The simple local architecture, not replicated, seems to be the fastest option in all cases, with an average of 15,957 microseconds.

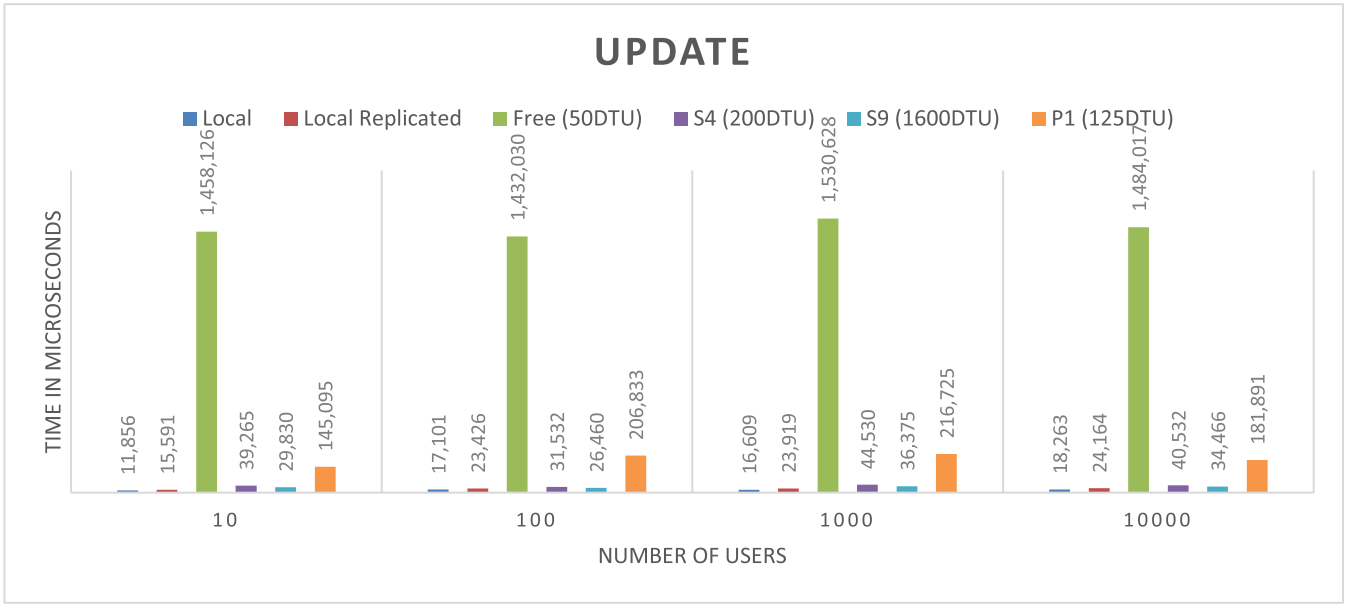

FIGURE 4. Update statement experimental results.

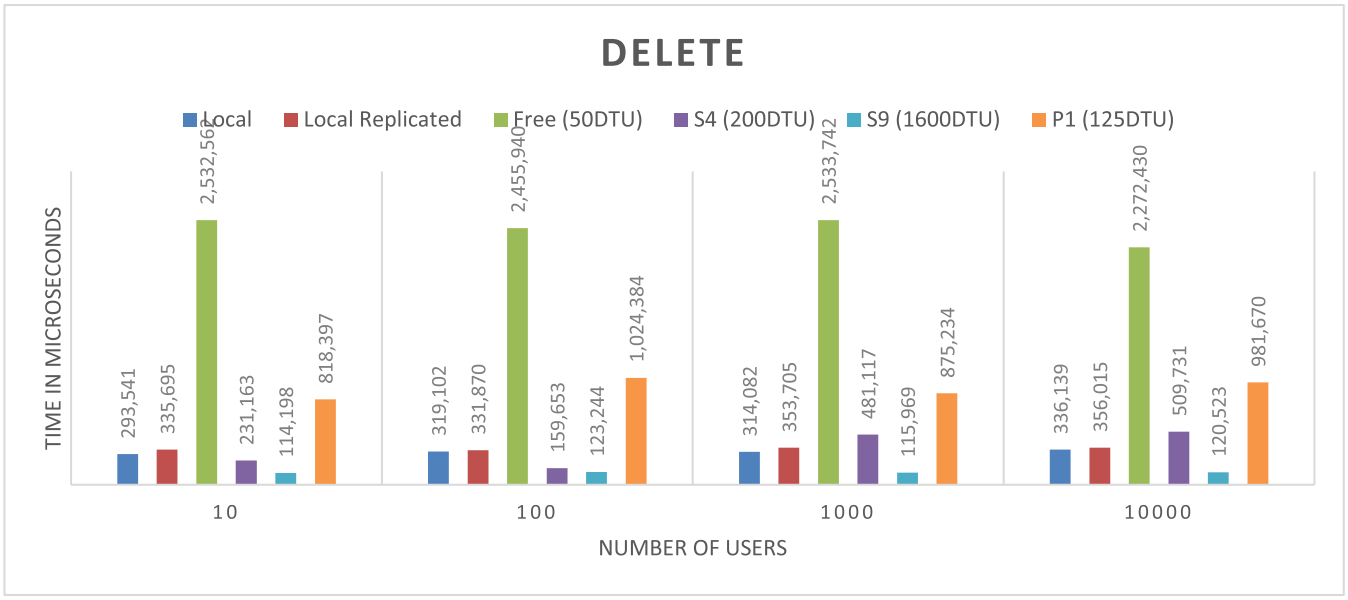

FIGURE 5. Delete statement experimental results.

In case of **DELETE** statement, the procedure code is deleting the first 3 data entries that match the constraints from the WHERE clause:

```
DELETE TOP (3) p
FROM Produse.Product p
INNER JOIN Preturi.TVA t ON t.TVAID = p.TVAID
WHERE p.Available = 1
AND p.ProductTypeCode like '%PROD%'
AND t.TVAID = 3
```

The **DELETE** statement is also a CPU intensive operation that requires a lot of processing units. As it is shown in Figure 5, Azure S9 is the best approach for DELETE statement. In this case, for all number of users that we used, the S9 is the fastest approach with an average of 118,483 microseconds. Next to it is on-premises architecture with an average of 330,018 microseconds and Azure S4 with an average of 345,416 microseconds. On-premises architecture seems to be a little bit faster for a big number of users, and Azure S4 seems to be the most appropriate to on-premises. On the fourth place we can place the Azure Premium 1, with an average of 900000 microseconds. The last place is Azure Free pricing tier, and this is because of the small number of DTUs. It obtained an average of 2.3 seconds.

IV. PRICE COMPARISON

To make this comparison as accurate as possible we decided to include a small list of prices for both on-premises equipment and software and for Azure pricing tiers that we used.

TABLE 2. On-premises hardware and software costs.

Equipment/Software	Price (€)
Windows Server 2012 R2	1,000.00 € x 2
Server + Hardware Stuff	7,000.00 € x 2
TOTAL	16,000.00 €
TOTAL COST deducted in 3 years, by monthly fee	16,000.00 / (12 * 3) ≅ 444.00 € / month
Internet provider	215.00 € / month
Sys Admin salary (1 hour / day)	250.00 € / month
TOTAL / month	909.00 € / month

Table 2 presents the equipment needed for on-premises architecture (hardware and software) that are close to our testing environment together with approximate prices attached. For local architecture, the one with replication server in considered. In the end, after we covered the entire costs with equipment, we will pay an amount of **909.00 €/month** together with Sys Admin salary (one hour per day), and internet (bare necessities).

TABLE 3. Azure tiers monthly costs.

Pricing tier /stuff	Price (€)
Free	0.00 €
Standard 4 (200DTU/250GB)	126.50 €
Standard 9 (1600DTU/250GB)	1,012.02 €
Premium 1 (125DTU/500GB)	392.13 €

For Azure approach, Table 3 presents the prices per month for each pricing tier that we used in our tests. For Azure prices we must specify that we also need to use an internet provider, so we will keep the same cost as the on-premises scenario, that is 215.00 €/month.

Based on estimations form Table 2 and Table 3, the most expensive option when using cloud platform, is Azure S9, where the average cost per month would be: Internet provider (215.00 €) + Azure S9 (1,012.00 €) = **1,227.00 €/month.**

According to the obtained results, a possible replacement for Azure Standard 9 could be Azure Standard S4. This pricing tier obtained a good time in our tests and we can say it is a good option for us if we want to decrease the costs. Here the entire costs per month would be: Internet provider (215.00 €) + Azure S4 (126.00 €) = **341.00 €/month.**

Based on our comparison, we can observe that cloud architecture is generally more expensive than on-premises architecture, when we refer to high performance and speed, for example, when considering Standard S9 pricing tier with a total cost of 1,227.00 €/month. But, if a balance between costs and performance is needed, standard S4 pricing tier could represent a very good option, with lower total cost than of on-premises architecture (341.00 €/month, instead of 909.00 €/month) but with much better performance, in some cases comparable with upper pricing tiers.

Consequently, in terms of costs, we can easily see that instead of using on-premises architecture at a higher price, we can use the cloud architecture, where we will have smaller costs and, when properly chosen, better performance.

V. CONCLUSION

The purpose of making this study was to answer the question that was placed at the beginning of this paper, namely: Why cloud? By developing the Restaurant app, we intended to see the differences between on-premises storage and cloud storage. Consequently, we developed this app to be capable to work with data that are coming from a local server and data from cloud. Based on this app, several tests were carried out for each basic SQL operation, using some real-application queries, and the obtained results were further analyzed. During this study we reviewed the existing cloud platform alternatives. We discovered that on the market there are many companies that allow users to access their space and applications from cloud and develop their own network of virtual servers or custom applications.

We can conclude that not in all circumstances the cloud approach brings better performance. In some cases, response time results show that the cloud database performance is poor by comparing to the traditional one, especially when the free cloud tier is implied. As it was shown in the presented results, moving into the cloud represents better choice in terms of performance than on-premises architecture when large number of users are implied and when appropriate (means, higher) price tiers are chosen.

However, we can see that, in some cases, the difference between the two approaches is measuring milliseconds; consequently, cloud computing with flexible pricing models could present generally the best solutions to be able to benefit for the whole power and convenience of the cloud: usability, scalability, reliability, security and, not in the last, price.

There is another way to store data, using Private hosting where you can buy space and servers to run the apps, but in this way, you will not get the same stability as cloud, because also there will be someone who will take care about servers, infrastructure and other stuffs which involves more money and more time spent.

However, each approach of storing data has advantages and disadvantages, but as we can see the whole world is dependent on internet. All modern apps and all big companies like Microsoft, Google, Facebook, Amazon or others are using cloud to store data because it is comfortable to access it any time and from anywhere, and most importantly, they are always safe. This is the future of storing data, no more hardware to buy and no more systems to administer because all of these are just a mouse click away.

REFERENCES

[1] B. Butler. 10 of the Most Useful Cloud Databases. Network World. Jun. 2014. [Online]. Available: http://www.networkworld.com/article/2162274/cloud-storage/cloud-computing-10-of-the-most-useful-cloud-databases.html.
[2] T. Whitney. Guide to Universal Windows Platform (UWP) Apps. May 2016. [Online]. Available: https://msdn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide
[3] M. Patterson. What is an API, and Why Does it Matter?. Apr. 2015. [Online]. Available:http://sproutsocial.com/insights/what-is-an-api/
[4] C. Rabeler. SQL Database Options and Performance: Understand What's Available in Each Service Tier. May 2016. [Online]. Available: https://azure.microsoft.com/en-us/documentation/articles/sql-database-service-tiers/rnd=1
[5] A. Marathe *et al.*, "A comparative study of high-performance computing on the cloud," Presented at the HPDC, New York, NY, USA, Jun. 2013, pp. 239–250. [Online]. Available: http://www.cs.arizona.edu/dkl/Publications/Papers/hpdc13.pdf
[6] R. R. Expósito, G. L. Taboada, S. Ramos, J. Touriño, and R. Doallo, "Performance analysis of HPC applications in the cloud," *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 218–229, Jan. 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X12001458
[7] C. Gyorödi, R. Gyorödi, and R. Sotoc, "A comparative study of relational and non-relational database models in a Web-based application," *Int. J. Adv. Comput. Sci. Appl.*, vol. 6, no. 11, pp. 78–83, 2015, doi: 10.14569/IJACSA.2015.061111.
[8] D. Ghoshal, R. S. Canon, and L. Ramakrishnan, "I/O performance of virtualized cloud environments," presented at the 2nd Int. Workshop Data Intensive Comput. Clouds (DataCloud-SC), Seattle, WA, USA, 2011, pp. 71–80. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.232.4019&rep=rep1&type=pdf
[9] S. Narula, A. Jain, and M. Prachi, "Cloud computing security: Amazon Web service," presented at the IEEE 5th Int. Conf. Adv. Comput. Commun. Technol., Feb. 2015, pp. 501–505, doi: 10.1109/ACCT.2015.20.
[10] G. Carutasu, M. A. Botezatu, C. Botezatu, and M. Pirnau, "Cloud computing and windows azure," presented at the ECAI-Electron., Comput. Artif. Intell. Int. Conf., Ploieşti, Romania, Jun./Jul. 2016, pp. 1–6, doi: 10.1109/ECAI.2016.7861168.
[11] M. Abourezq and A. Idrissi, "Database-as-a-service for big data: An overview," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 1, pp. 157–177, 2016, doi: 10.14569/IJACSA.2016.070124.
[12] W. Al Shehri, "Cloud database database as a service," *Int. J. Database Manage. Syst.*, vol. 5, no. 2, p. 1, 2013. [Online]. Available: http://airccse.org/journal/ijdms/papers/5213ijdms01.pdf, doi: 10.5121/ijdms.2013.5201.
[13] S. Jain, "Comparative study of traditional database and cloud computing database," *Int. J. Adv. Res. Comput. Sci.*, vol. 8, no. 2, pp. 80–87, 2017. [Online]. Available: http://www.ijarcs.info/index.php/Ijarcs/article/viewFile/2935/2918
[14] A. Litchfield, A. Althwab, and C. Sharma, "Distributed relational database performance in cloud computing: An investigative study," *J. Inf. Technol. Manage.*, vol. 29, no. 1, pp. 16–46, 2018. [Online]. Available: https://www.researchgate.net/publication/324068566_Distributed_Relational_Database_Performance_in_Cloud_Computing_An_Investigative_Study
[15] A. Thakar, A. Szalay, K. Church, and A. Terzis, "Large science databases–are cloud services ready for them?" *Sci. Program.*, vol. 19, nos. 2–3, pp. 147–159, 2011, doi: 10.3233/SPR-2011-0325.
[16] S. D. Bijwe and P. L. Ramteke, "Database in cloud computing-database-as-a service (DBaas) with its challenges," *Int. J. Comput. Sci. Mobile Comput.*, vol. 4, no. 2, pp. 73–79, 2015.
[17] K. Chitra and B. J. Rani, "DES: Dynamic and elastic scalability in cloud computing database architecture," *Int. J. Adv. Comput. Sci. Appl.*, vol. 5, no. 1, pp. 173–175, 2014, doi: 10.14569/IJACSA.2014.050124.
[18] A. Sleit, N. Misk, F. Badwan, and T. Khalil, "Cloud computing challenges with emphasis on Amazon EC2 and windows azure," *Int. J. Comput. Netw. Commun.*, vol. 5, no. 5, pp. 35–44, Sep. 2013, doi: 10.5121/ijcnc.2013.5503.
[19] Jan. 2018. *Database Transaction Units (DTUs) and Elastic Database Transaction Units (eDTUs)*. [Online]. Available:https://docs.microsoft.com/en-us/azure/sql-database/sql-database-what-is-a-dtu
[20] (Mar. 2017). *Tutorial: Configure Replication Between Two Fully Connected Servers (Transactional)*. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/replication/tutorial-replicating-data-between-continuously-connected-servers?view=sql-server-2017
[21] (Nov. 2018). *Set Up Disaster Recovery for Azure VMs to a Secondary Azure Region*. [Online]. Available: https://docs.microsoft.com/en-us/azure/site-recovery/azure-to-azure-tutorial-enable-replication

ROBERT GYŐRÖDI received the M.Sc. and Ph.D. degrees in computer science from the Politehnica University of Timisoara, Romania, in 1995 and 2001, respectively. Since 2009, he has been a Full Professor of computer science with the Department of Computer Science and Information Technology, University of Oradea, Romania. He has over 15 years of IT consulting experience. He has authored/co-authored ten books, and over 80 published papers, many of them in journals and international conferences. His current research interests include a combination of image processing, artificial intelligence, data mining, and database management systems subjects. He is an IEEE Member and the Acting Chair of the Microsoft Dynamics Academic Alliance EMEA Council.

MARIUS IULIAN PAVEL is currently pursuing the M.Sc. degree with the Department of Computer Science and Information Technology, University of Oradea, Romania, with a focus on cloud computing and database management systems subjects.

CORNELIA GYŐRÖDI received the B.Eng. degree in automation and computers and the Ph.D. degree in computer science, in the area of data mining, from the Politehnica University of Timisoara, in 1994 and 2003, respectively. She has been a Professor with the Department of Computer Science and Information Technology, Faculty of Electrical Engineering and Information Technology, University of Oradea, Romania, since 2009. She is currently a local Coordinator of the Oracle Academic Initiative, and participated at different specializations in this area, organized by Oracle Romania. Her current research interests include database management systems, knowledge discovery in databases, and data mining techniques. She has authored/co-authored 11 books and over 85 publications in the aforementioned fields.

DOINA ZMARANDA received the B.S. degree in automation and computers and the Ph.D. degree in computer science, with a focus on real-time control systems, from the Politehnica University of Timisoara, in 1990 and 2001, respectively. Since 2009, she has been a Professor with the Department of Computer Science and Information Technology, University of Oradea, Romania. She has authored/co-authored over 60 publications in the field of computer science. Her research interests include real-time application development, Web and cloud application development, concurrent programming, and system's modeling and simulation.