An Extended Evaluation of Schema Issues Advisor in the Azure SQL Database

Dejan Dundjerski, Stefan Lazić, Milo Tomašević, and Dragan Bojić

***Abstract* — The analysis of the telemetry data in Azure SQL database has revealed that most of the problems affecting the customers are due to the schema inconsistency errors. An assumption has been made that direct e-mail notifications sent to the customers about the current problems could significantly shorten time to resolve them. The prospective benefits are first validated by sending e-mail recommendations manually. Then, in an iterative way, the schema issue advisor that detects anomalies and automatically sends the appropriate notifications to the customers is implemented on top of Azure resources. Finally, an extended evaluation has confirmed the expected benefits of direct communications with the customers.**

***Keywords* — Azure SQL database platform; Schema issue advisor; Feature validation and testing; Cloud services.**

I. Introduction

Introduction of the cloud environment in recent years has provided new possibilities and advantages to its customers. Existing on-premise solutions have been massively migrated to the cloud due to lower cost of maintenance [1], [2]. In addition, companies which used to build complex software solutions that were installed on-premise with the development cycles of several years, now develop, deploy and publish new versions of software with a monthly pace [3]. This kind of approach allows the companies to validate the design decisions and get feedback from the customers much faster.

Paper received April 7, 2018; revised October 30, 2018; accepted November 11, 2018. Date of publication December 25, 2018. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Miroslav Lutovac.

This paper is a revised and expanded version of the paper presented at the 25th Telecommunications Forum TELFOR 2017 [5].

Dejan Dundjerski – School of Electrical Engineering, University of Belgrade, Bulevar Kralja Aleksandra 73, 11020 Beograd, Srbija (e-mail: dd145019p@student.etf.rs). Microsoft Development Center Serbia, Spanskih Boraca 3, 11000 Belgrade, Serbia (e-mail: dejandu@microsoft.com).

Stefan Lazić – School of Electrical Engineering, University of Belgrade, Bulevar Kralja Aleksandra 73, 11020 Beograd, Srbija (e-mail: ls163150m@student.etf.rs). Microsoft Development Center Serbia, Spanskih Boraca 3, 11000 Belgrade, Serbia (e-mail: stlazi@microsoft.com).

Milo Tomašević – School of Electrical Engineering, University of Belgrade, Bulevar Kralja Aleksandra 73, 11020 Beograd, Srbija (e-mail: mvt@etf.bg.ac.rs).

Dragan Bojić – School of Electrical Engineering, University of Belgrade, Bulevar Kralja Aleksandra 73, 11020 Beograd, Srbija (e-mail: bojic@etf.bg.ac.rs).

At the same time, a wide diversity of ways how the software in the cloud can be used and incorporation of trending technologies such as machine learning and expert systems enable scenarios that were not possible before. This particularly applies to possibilities of the automatic performance troubleshooting and automatic tuning of the relational databases which were initially introduced in the Azure SQL database [4]. More generally, it refers to detection of any problem that can arise and affect the customer workload, availability and function of the end-service and application which stores its state into a relational database. Collecting the telemetry data that tracks the exceptions and errors as they occur in the customer database enables the detection of such potential problems. It was also recognized that the customers are usually unaware of the problems for a relatively long period of time. Therefore, our intention was to propose an improved advisor that would automatically generate the appropriate notifications.

In this way, the users would be instantly informed about recently detected errors in the database, particularly about those related to the schema inconsistencies which directly affect functioning of an end-service or application. The proposed approach of automatic, prompt alerting the users should improve the time to detect and mitigate schema inconsistencies and issues.

Before realization of this idea, a general proof of concept was necessary. Validation of the proposed feature was performed by manual notifications. After that, the main challenge was how to shorten the development cycle from the idea to realization. An improved, agile process of the feature development through integrated implementation, testing and evaluation in a continuous loop was carried out. An end-result should be an improved schema issue advisor based upon the customer feedback.

This is an extended version of the paper [5] with further evaluation data and some additional discussion and explanations. Section 2 briefly describes the Azure SQL database platform. Telemetry pipeline and database advisors are especially covered, as they are important components of the platform in the context of this work. Section 3 elaborates the statement of the problem observed in the Azure SQL database service. Motivation and validation of the proposed solution is described in Section 4. Section 5 gives an overview of the implementation process and continuous testing of the feature. Evaluation is presented and discussed in Section 6. Conclusions and highlights of the prospective future work are brought in Section 7.

II. AZURE SQL DATABASE

One of the first database-as-a-service solutions is Microsoft Azure SQL cloud-based database [3]. It is designed and built as a highly available and reliable system. Such a system alleviates the need for maintaining the servers that host SQL servers. At the same time, having a database hosted in the Azure SQL DB environment, ensures the latest version and new features of the SQL server before those are delivered as a standard on-premise version. These kinds of services provide telemetry that tracks resource utilization, encountered errors and other useful and relevant information for a single database.

A. Telemetry

The Azure SQL database hosts millions of databases across the globe at the same time. Quite alike to the flight data recording box in a plane, all important parameters of the database are collected and recorded [6]. In our approach, it is of vital importance to track the information about errors and exceptions that continuously occur inside the Azure SQL database service.

Telemetry that is being recorded, based on importance of data, are taken with different frequency periods. The most important data are collected with the granularity of seconds but the retention period for such data is lower. Less variable data, such as query store statistics, are stored with the granularity of minutes. The information that is not changed that frequently, is sampled a couple of times per day. Telemetry is being stored in a structured way in both cold and hot storage in compliance with General Data Protection Regulation [7]. Hot storage is used for scenarios like alerting, where it is very important to have data with a low lag. Querying of hot storage should be completed in a restricted amount of time providing near real-time results. On the other side, cold storage contains much more data whose retention period is longer, so it is often used for post-processing and various ad-hoc analyses. Due to the amount of data being stored, querying of cold storage is usually slower and cannot be used for near real-time processing.

B. Database advisors

Collecting the telemetry data from a very large number of different databases allows an important possibility. Even though numerous customers have different workloads, still typical patterns can be extracted and recognized as far as the usage of the database is concerned. By analyzing the telemetry data, those different workloads could be grouped according to their workload thumbprints. Once the specific thumbprint or behavior is detected, there is a possibility to understand it and to conduct better troubleshooting of this specific case. Understanding different cases enables the platform to provide an insight about certain activities to the user. With the detection of specific issues, automatic tuning of the database is also possible for the cases where the appropriate corrective actions are available.

This approach can also catch issues caused by a defect in the newly deployed application or changes in the database schema. Consequently, early detection of these issues allows the customers to be focused on fixing the actual issue instead of building the complex monitoring and alerting logic. This approach saves a significant time for the customers and enables them to concentrate on other issues closely related to their business. Two existing advisors that follow this kind of approach are Database Index and Parameterization Advisor [4].

III. PROBLEM STATEMENT

By analyzing the telemetry from the Azure SQL database service, we concluded that various errors occurring in the service could be classified into different groups. The largest class, that covers about 25% of the observed cases, consists of the errors caused by the database schema inconsistencies. These issues usually happen either due to changing the application logic or due to a change in database schema. When the customer changes the application logic, it usually results in a set of queries targeting non-existing tables, columns and stored procedures. Otherwise, when an inconsistent state comes from the database side, it usually happens when the schema was unintentionally modified or when the deployment process between the application and the database is not orchestrated correctly. These issues affect the proper functioning of end-service or application which employs a relational database to preserve its state. The size of impact depends on the business logic and part of the database or application that is being modified.

Other classes that represent the errors not related to the schema inconsistencies are not that large since they cover a wide spectrum of issues. Therefore, the first iteration of actual implementation will be limited to the largest group of errors. Nevertheless, providing an insight about other errors might be also beneficial for the users, so the validation of concept will include those errors as well.

Focusing on the schema inconsistencies revealed another useful insight obtained from the telemetry. Once the users change their application or schema of the database, they don't become aware of the problems that such a change caused immediately. Either due to the lack of telemetry on the application side or bad programming practice, when the errors and exceptions are caught without an appropriate report, the usual time to detect an error and resolve it from the customer perspective was about 2 to 3 days. This represents a potential problem that could cause the improper functioning of the application or its complete failure, depending on the application and business logic.

Related work done in [8] relies on the concept of machine learning to assist in the detection of service issues. This study is focused on the machine learning models for classification of different issues while less focus has been given on the interaction with the customer about the detected problems. By analyzing the different systems and solutions on the global market [9], it has been concluded to the best of our knowledge, that there is no end-to-end solution that adequately addresses the stated problem of the schema issues which affect the business and application logic. Specifically, other cloud service providers use the telemetry data to analyze and recognize different patterns as a base for continuous improvement and optimization of the service they are hosting. Our goal is to go one step further and use the telemetry data not just to improve the system, but to improve the customer experience as well.

IV. Iterative Validation of the Concept

The first iteration had to validate that observed pattern that notification about the errors and specifically schema inconsistencies can provide benefit to the customers before continuing with actual implementation. The intent was to estimate to which extent detecting SQL errors and informing customers about the caused problems helps.

Prototype of the model for detection of error anomalies was based on a simple query over the hot telemetry data source. It helped us to detect the databases with the highest error counts in the current hour compared to the average error count per hour during previous 5 days. The baseline period of 5 days was dictated by the limitations of the hot telemetry data source. In such a way, top ten databases with the highest error counts were identified and then investigated manually.

For each of ten databases, a dominant error with the highest count of occurrence was recognized and investigated. In databases where Query store feature [10] of the SQL server was enabled, investigation included an additional step where we checked if there were newly introduced queries, based on the query hash. After that, the customer was informed with a manually sent e-mail message about the observed issue including the number of errors, predicted trend and potential clues how it should be further examined from application standpoint. Initial reply rate was quite high – about 60% of customers replied to the email. They were satisfied with the notification and found out such information very useful. This was encouraging enough to continue with the next iteration of the investigation and validation.

Further iterations included more than top ten databases, and at the same time, different groups of detected errors were investigated as well. Different groups of errors had different reply rates, but one group stood out clearly throughout the entire validation period. This was the group of errors related to the schema inconsistencies.

During the validation process, manually sent e-mail messages and their templates evolved. Initially the e-mails included only a basic set of information. Since the customers asked for additional information, we added a more clear and detailed insight in further iterations. This kind of feedback loop helped us to iterate with initial solution much faster during the validation. After a couple of iterations, by means of direct communication with the customers we verified our assumption that, without informing the users, it takes between 2 and 3 days to resolve the schema inconsistency. Additional feedback from the customers was collected during this phase and taken into consideration during the implementation of the final solution.

V. Implementation and Continuous Testing

After we proved the concept of alleviating the query failures due to the issues of inconsistency between application queries and database schema, the enhanced SQL Database Schema Advisor was implemented [4].

A. Implementation details

The high-level diagram of the implemented workflow is presented in Fig. 1.

Given the fact that every single error that happens in a database is stored separately, data encompassed in the implemented model is preprocessed to obtain the count of errors in a 15-minute interval classified by actual error messages and the database where the error occurred. Actual error message is chosen as an additional classification criterion to differentiate between errors with the same error code but with different error messages. For example, multiple different queries might target more than one different non-existing table, columns or other database objects. Errors that happen in that case have the same error code, but corresponding error messages are different.

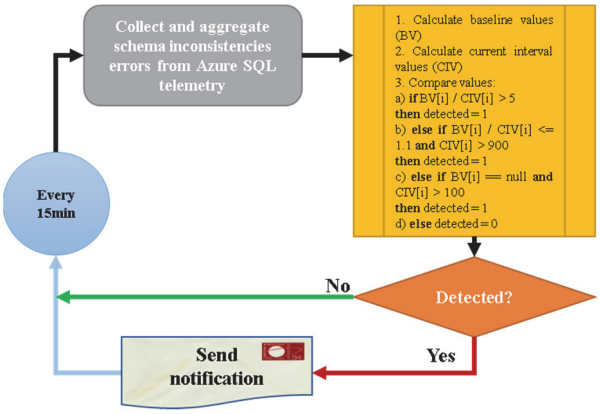

Fig. 1. Implemented workflow.

Collecting data and their preprocessing in 15-minute intervals are done for every database in the Azure SQL Database service. The database schema issue advisor tracks only errors caused by the schema inconsistencies. The errors enlisted below cover 95% of all schema inconsistency errors:

- 201: Procedure or function '<ProcName>' expects parameter '<ParamName>', which was not supplied.
- 207: Invalid column name '<ColumnName>'.
- 208: Invalid object name '<ObjectName>'.
- 2812: Could not find stored procedure '<ProcName>'.
- 8144: Procedure or function '<ProcOrFuncName>' has too many arguments specified.

The rest 5% of errors caused by the schema inconsistencies are very rare and do not produce significant anomalies. In order to reduce the amount of data being collected and processed, the focus was given only on the enlisted, most frequent errors.

This paper proposes an anomaly detection algorithm which is based on comparison between counts of collected schema inconsistency errors during the baseline period and monitoring period. A snapshot of the error counts in relevant time frames is presented in Fig. 2.

The baseline period (green area) consists of 60 2-hour intervals (5 days in recent history) which is used as a reference period for comparison with the current 2-hour monitoring period (orange area). For a fair comparison to total error count in the monitoring period, the error count in the baseline period is normalized to a 2-hour time interval. Drift period (white area) consists of 12 2-hour intervals and

it serves for the long-lasting anomalies to settle, in order not to affect the baseline period error count when the anomaly reoccurs soon after it is being resolved from the customer perspective.

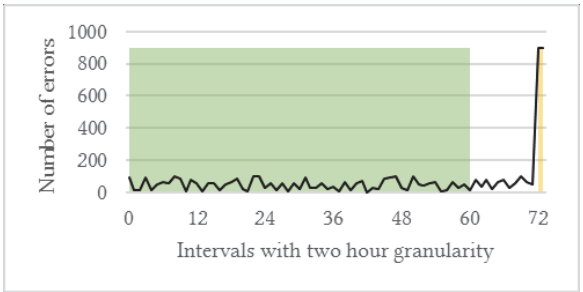

Fig. 2. Trendline of the collected error counts in baseline (green), drift (white) and monitoring (orange) periods.

The 2-hour interval duration was taken after a careful analysis over telemetry data in order to increase the probability that an anomaly lasts for some time and to avoid transient issues which are not as significant for the customers. In this way, the relevance of the insight is increased, and a customer is relieved from less relevant notifications. Examined telemetry data included detected schema inconsistency errors over a six-month period on all databases in the Azure SQL database service. During observed period, 7% out of 1 million customer databases were producing 700 million errors per day. Durations of the baseline period (5 days) and the drift period (1 day) were taken after we examined the size of telemetry data considering the total number of databases in the service and the total number of recommendations that the anomaly detection algorithm could produce.

Once the monitoring is started, the anomaly algorithm is executed every 15 minutes, and the baseline, drift and monitoring periods are shifted in 15-minute increments simultaneously. When a schema inconsistency anomaly is detected for one database, this anomaly is further monitored and tracked continuously. After it is being resolved from the customer perspective, it will take another 2 days until the system marks this anomaly as resolved and stops monitoring it. Duration of this cool down period was also taken after careful analysis over the telemetry data. The results indicated that the average time to resolve the schema inconsistency issue is 2 days. If the schema inconsistency does not reoccur during the cool down period, it is regarded as completely resolved. Otherwise, if it reoccurs again, the system continues to monitor it until the customer completely fixes the issue. The cool down period with continuous monitoring for the set of schema inconsistencies was introduced in order to minimize the probability to miss the issue.

Based on the comparison between the error count during monitoring period and the error count during baseline period, three characteristic types of the anomaly behavior can be recognized.

A sudden increase of the error count is a rather frequent case when the number of errors in the monitoring period compared to the baseline period is higher than a certain threshold (5x increase). Such a situation is illustrated in Fig. 2. In this case, it is very likely that something was changed either on the database or application side. Previously, before introduction of the schema issue advisor, after a certain period (in this case 6 days) such an issue would not be detected and customer would not be notified.

Another case is observed through a constantly high number of errors. This happens when there is no sudden increase of the error count, but it is steadily above a specific threshold. In this case, the threshold is configurable, and it is set to one error per second. This scenario is usually not that important for the customer, but if we detected the moment when this anomaly appeared, such an anomaly would fall into previous category and the appropriate notification would be more helpful to the customer.

The last observed case represents a sudden increase in the number of errors for new databases non-existent during the baseline period. It can happen that newly created databases do not have the baseline. Such databases existed from the service and telemetry points of view only in the recent monitoring intervals. In this case, the errors usually happen when customer deploys a new schema, tests its effects and experiences potential defects both in the database schema design and in the application design. Despite the fact that there is a relative increase in the number of errors, it might happen that the absolute number of errors is relatively small, so this information would not be that useful.

The Schema issue advisor which preserves preprocessed telemetry about the errors, implements the anomaly detection model and keeps the detected anomalies for each customer database is based on one instance of the Azure SQL Database. Portal and e-mail notification channels already existed in the expanded platform, so the model database was added as an additional data source to the existing notification channel. Notification channel for its consumers is exposed as an internal API which receives information about the customer to which the notification will be sent, type of the notification, template that will be used in the notification message and arguments that fill in the template. The Azure Data Factory service [12] as an easy-to-use scheduling platform for data collection, data processing scheduling and data transferring was also reused for this purpose.

B. Continuous testing

After the proposed concept was implemented on top of existing technologies that the Azure stack offers, continuous testing and evaluation were carried out. In the service world, it is very important to know what is happening with the provided service, so the continuous validation is unavoidable. Once the customers start using the schema issue advisor, if there is a problem with their database, they will rely on notifications about the issue from the Azure SQL database platform in the future as well. It also might happen that some part of the pipeline does not work properly which prevents sending the notifications to the users. If a customer detects a problem with its database without being informed from the advisor, the customer would notice and report the lack of the notification that it used to receive. Since not all components on top of which schema issue advisor was built are in the direct control of

the schema issue development team, the need for continuous system verification is even more required. Another requirement in case of the pipeline and system failure is to exactly know which part of the pipeline failed so that the engineers can investigate it relatively quickly and fix the exact source where the issue occurred.

For this purpose, a dedicated test task called runner was implemented by the schema advisor development team. The periodic test executed by this runner takes one production Azure SQL database that serves for testing purposes. It repeatedly and continuously executes the following steps with the desired frequency:

- Runs a set of test queries against the test database that generate the schema inconsistency errors which triggers the anomaly detection algorithm.
- Waits for a predefined period of time, for the telemetry to become available and for the issue to be detected by the anomaly detection algorithm.
- Checks whether the raw telemetry data source has the information about the errors that were generated. If this check failed, it means that there is a problem in the platform itself, since the telemetry does not flow.
- Checks whether the anomaly detection model detects the set of errors and aggregates the data in the correct manner. If this check failed, it means that there is an issue with the schema issue advisor model.
- Checks whether the created anomaly in the previous step reaches the post-processing phase. If this check failed, it means that there is an issue in the existing advisor framework that schema issue advisor uses.
- Checks whether the appropriate portal notification appears and whether the system sends an e-mail to the customer. If this check failed, an issue between the advisor and notification framework is detected.

This model of continuous testing and validation can be applied for any kind of pipeline or the process that includes different mutually connected components. This is a widespread design pattern followed by every cloud solution based on the multiple services. In such a system, in order to discover where the actual issue comes from, it is desirable to have a clear separation between different components or services in order to make further debugging and investigation process easier.

VI. Evaluation

Implemented schema issue advisor during the period of 10 months detected more than 200,000 anomalies. These anomalies were detected on different production databases utilized by different customer workloads and applications. Initial implementation of the advisor framework displayed this kind of information only on the Azure portal without direct e-mail notifications to the customers. E-mail notifications to the customers were added in a later version. In the last 5 months when e-mails with recommendations were sent to the users, their number was more than 100,000 (slightly more than the number of portal notifications only).

Regarding the average time to resolve the issue, our results show an improvement from 19 hours and 52 minutes, for portal notification only, to 12 hours and 54 minutes for recommendations that included e-mail notifications, which is a significant overall improvement of 54%. If we consider only the recommendations for the users (9.4% of all schema advisor users) that saw the issue through portal notification first, and then saw the issues through e-mail or portal after the e-mail was introduced, the average time to resolve was significantly reduced from 2 days and 45 minutes to 8 hours and 31 minutes, which is an improvement of more than 3x times. Fig. 3 shows the distribution of the counts of the resolved schema issue recommendations according to the time to resolve the issue for the customers that saw the recommendations through portal notification (with and without e-mail notification) that lasted less than 2 days. There is a noticeable pile of recommendations without e-mail notification (portal only) for which the time to resolve is almost 24 hours (gray x). On the contrary, visible piles of recommendations with e-mail notification can be recognized on the same histogram for the times below 2 hours, around 6 hours, between 8 and 16 hours, and 20 hours (black crosses). As for the issues resolved in less than 2 hours, more of them were fixed (about 2000 more or 4.7% of the observed dataset) when e-mail notification was sent compared to those fixed when the portal notifications was sent. This is an additional indication that the average time to resolve the schema issue is reduced. However, if we consider the issues resolved between 2 and 6 hours, the issues resolved with portal notifications only slightly prevail (about 380 more). It can be explained by the fact that the active users of the schema issue advisor usually rely on this advisor to catch their issues after the deployment. So, in case of such an inconsistency they perform the rollback of the problematic deployment immediately. If the issue is not that severe, it takes up to one working day (between 6 and 8 hours) to fix the issue and deploy an updated version.

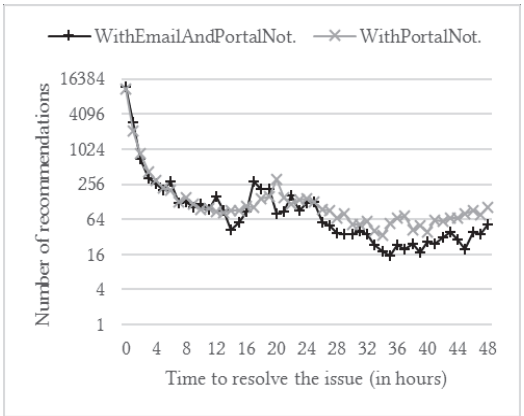

Fig. 3. The time to resolve the schema issues in hours with and without e-mail notifications in a period of 2 days.

Fig 4 presents the histogram with the counts of resolved schema issue recommendations with the time to resolve the issue of not more than 30 days. The number of the resolved recommendations in less than 24 hours is slightly higher for the recommendations with e-mail notification. As for schema inconsistency resolutions that lasted more than a day, those attributed to the recommendations displayed

through portal notification only, are more frequent. Therefore, it is very probable that they were not noticed by the customers at the time when the issue occurred, increasing the latency of fixing.

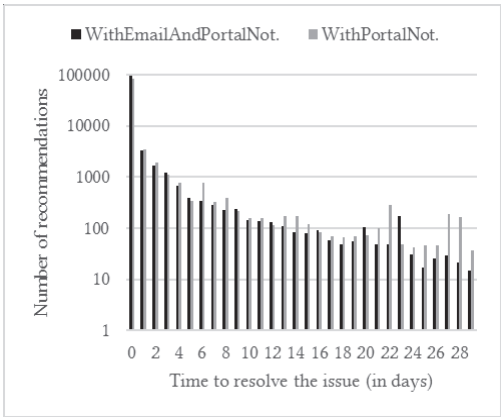

Fig. 4. Histogram that represents the time to resolve the schema issues with and without e-mail notifications in a period of 30 days.

Finally, it would be interesting to group previous results based on the application type with special attention to business and mission critical application. Unfortunately, given the high number of impacted databases, it was not possible to investigate each workload separately and, in that way, do the further categorization of the results. Example of usefulness of the schema issue advisor can be illustrated by one interesting case detected during the evaluation. It was a problem with an application which exhibits IoT (Internet of Things) scenario. The devices were scattered without the possibility to know whether a device works or not if it encounters an error. A change in the database schema was made when the stored procedure for activation of the device was modified. All devices started hitting the error 201 – a specific parameter was not supplied. Due to lack of telemetry and alerting mechanisms on the application side, recommendation and notification produced by the schema advisor for this case was very useful since it caught the real problem in an important scenario which was not detected.

VII. Conclusion

An advanced feature like database advisor in the Azure SQL database enables an analysis of the users' workloads and anomalies that inevitably happen in the software development. With this approach, we have tried to improve delivered service to the customers by deep profiling of the errors and direct alerting of them through e-mail notifications. After manual validation, implementation and iterative evaluation of the advanced schema advisor concept was conducted on top of the Azure platform.

The development of the improved schema issue advisor is an example of a modern agile development process in the environment of cloud services. A rapid iterative implementation process was supported by simultaneous and continuous testing. The customer validation and feedback helped us to reduce the development cycle of the new feature.

The evaluation confirms a significant decrease in time to resolve the issue with a recommendation sent through e-mail notification for the customers that eventually saw the portal notification, especially for times to resolve the issue within two hours. The response rate during the testing phase was higher than the response rate after the schema issue advisor was deployed. This was explained by the fact that customers usually neglect template-formatted e-mails from the platform. That was the reason why a direct contact with more personalized e-mails acquired a higher response rate.

Future improvement could be oriented towards experimentation with personalized e-mails in order to attract the attention of the customers more. A more promising avenue could be to build an expert system which is based on more sophisticated models that could understand the customer workload and provide insightful recommendations.

References

[1] A. Dhiman, Analysis of on-premise to cloud computing migration strategies for enterprises. Ph.D. Dissertation. Massachusetts Institute Of Technology, USA, 2011.

[2] T. Boillat and C. Legner, "Why do companies migrate towards cloud enterprise systems? A post-implementation perspective", In *Proceeding of the 16th Conference on Business Informatics (CBI)*. IEEE, Geneva, Switzerland, 2014, pp. 102–109.

[3] P. Bernstein, I. Cseri, N. Dani, N. Ellis, A. Kalhan, G. Kakivaya, D. Lomet, D. Lomet, R. Manne, L. Novik, and T. Talius, "Adapting Microsoft SQL server for cloud computing", In *Proceeding of the 27th International Conference on Data Engineering*. IEEE, Hannover, Germany, 2011, pp. 1255–1263.

[4] Stein S, Rabeler C – Azure SQL Database advisor (2016). [Online]. Available: https://docs.microsoft.com/en-us/azure/sql-database/sql-database-advisor

[5] Dundjerski, D., Lazić, S., Tomašević, M. and Bojić, D., "Improving schema issue advisor in the Azure SQL database.", In *Proceeding of the 25th Telecommunication Forum (TELFOR)*, IEEE, Belgrade, Serbia, 2017.

[6] W. Lang, F. Bertsch, D. DeWitt, and N. Ellis, "Microsoft Azure SQL database telemetry", In *Proceeding of the Sixth ACM Symposium on Cloud Computing*. ACM, Hawaii, USA, 2015, pp. 189–194.

[7] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), (2018). [Online]. Available: http://eur-lex.europa.eu/eli/reg/2016/679/oj

[8] V. Nair, A. Raul, S. Khanduja, V. Bahirwani, Q. Shao, S. Sellamanickam, and S. Dhulipalla, "Learning a hierarchical monitoring system for detecting and diagnosing service issues", In *Proceeding of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Sydney, Australia, 2015, pp. 2029–2038.

[9] Amazon Web Services – Amazon RDS for SQL Server, (2016). [Online]. Available: https://aws.amazon.com/rds/sqlserver/

[10] Varga, S., Cherry, D. and D'Antoni, J., Introducing Microsoft SQL Server 2016, Microsoft Press, Redmond, USA, 2016.

[11] Lo S, Sreedhar P, Caro C – Introduction to Azure Data Factory Service, a data integration service, (2016). [Online]. Available: https://docs.microsoft.com/en-us/azure/data-factory/data-factory-introduction