Optimizing databases in the cloud based on performance and cost savings

FELIX NILSSON & DANIEL MIHAJLOVIC
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Optimizing databases in the cloud based on performance and cost savings

Felix Nilsson
fe4688ni-s@student.lth.se,
Daniel Mihajlovic
da7403mi-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Anders Klint & William Tärneberg

Examiner: Maria Kihl

August 2, 2022

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

As cloud service providers becomes more prevalent, so does questions related to cost efficiency of hosted resources. Payment models for cloud hosted resources tend to be either subscription based or pay-as-you-go for computing resources, in this case for compound metrics of CPU, Data IO and Log IO. In order to find necessary resource provisioning, previous methods have tended towards observing the utilization of these compound processing units and deciding based of some utilization threshold. This report aims to find metrics which can methodically present the state of a hosted database as well as suggest whether the current demand for resources is necessary. To do this, first a collection of metrics are found and analyzed in relation to how they present the database state. Then optimizations are done, as suggested by chosen metrics, to make the database efficient enough to require less processing power. From these experiments, it is clear that the behaviour of a less provisioned database come with indirect changes to query processing. As less memory is available, the reliance on reading data from disk becomes more prominent, leading to less efficient execution. As this occurs for many queries that run concurrently, wait times become a more dominant part of execution for overutilized resources. The execution plan for processing a query depends on the predicted impact estimation for available resources which can drastically change the nature of execution. If more resources are provided, and statistics related to previous resource provisioning is available, it is possible that some query performance degrades with more available resources. Essentially, the results of these experiments is that finding over-/under utilized resources depend on not only considering the utilization of resources but also wait times, and limitations to executions as a result of the available resources. The metrics suggested for optimizing the databases showed promising results, but the impact of the optimization remain hard to predict. This limits possibilities to choose changes that will lead to cost reductions under the limitations set by the cloud service provider.

Popular Science Summary - Characterizing the Provisioning of Computation Using Metrics

As different amounts of computing resources are used, execution changes to match. Not only does less computing power lead to lower capabilities to handle high workloads but the actual execution of functions often tend to become more resource demanding. In order to save costs for processing power, this is a consistent behaviour that has been observed and the key to understanding this is using metrics that can properly characterize the database under different provisioning.

Companies all over the world are changing from running their applications and computer infrastructure from their own organisation to using cloud service providers. This means that costs are based on how much of that cloud resource should be reserved for the company. The big problem that occurs then is, how much needs to be reserved? How can you be sure that the reserved resources are enough? For this article, these questions were put to the test for databases hosted in the cloud environment.

To solve any of these issues the first step is to determine how to observe resources, and their need for computing provisioning. This was done by gathering metrics regarding the utilization of database resources, the performance and the database efficiency. Using these three categories of metrics it is possible to evaluate how utilization and performance is currently related to current work, and database efficiency gives the opportunity to understand how resources can be spent more efficiently.

Using the metrics, and further exploring the impact of them, will give a methodology for evaluating resources without approximate thresholds and instead focus solely on the observable state of the database and how it is related to possible cost deficiencies in your cloud hosted infrastructure.

As costs, and available computing resources, are changed so is the effectiveness of execution. For a particular query the expected execution could reasonably be that it performs the same instructions irrespective of whether it requires 10% or 80% of available resources but that is not necessarily the case. With less memory provisioned the system requires more slow disk reads, instead of faster reads

from memory. When a query is run it is automatically optimized according to a execution plan that attempts to minimize cost. This plan can lead to different execution depending on how much resources the query demands. Finally, once less resources are provisioned it is common to see an increase in time spent waiting on acquiring computing resources. All these factors mean that execution for queries tend to cost more resources when less resources are provisioned.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

As cloud computing is growing more popular many companies migrate their data to the cloud. Solution such as database-as-a-service allows people to quickly set up a database without going through the hassle of acquiring the physical hardware, installation or configuration of the database.

Most cloud providers allow their consumers to scale the resources of the database in some capacity to handle various workloads. Requesting more resources come at an increased cost. In this case, a subscription model with access to a certain amount of computing provisioning in the form of a compound unit of CPU, Data IO, and Log IO. Alternatively it can work under a pay-as-you-go-model.

As a business wanting to adapt a cloud solution it can be hard to get the most of the service provided, while also spending as little as possible. Therefore it is important for companies to know if the resource being paid for is being utilized properly and if the performance is satisfactory for given resources.

1.1 Research Questions

The purpose of this thesis is to provide useful metrics regarding performance and utilization to be able to save cost for databases hosted by a cloud service provider. By considering certain metrics it should be possible to determine the right amount of computing resources needed to provide sufficient performance while keeping the cost low when using the cloud hosted resources. This focus resulted in two research questions:

RQ1. What indicators and metrics are important when measuring the efficiency of a cloud resource?

RQ2. How can under/over utilized resources be discovered?

1.2 Contribution

In the field of cloud computing related to cost savings it is often explored from the perspective of cloud hosting and minimizing the necessary resources in order to save costs. However exploring the effects of cost minimization using a cloud service provider is less studied. The effects of optimizing costs and usage of provided resources is a defining factor for saving costs for dynamic workloads using a cloud

infrastructure of hosted resources. By examining programs in relation to metrics for performance, utilization and efficiency we may be able to determine database resources that are in need of change for a better utilized system.

The purpose of this thesis is to develop a systematic approach to observe the performance of a database-as-a-service, in relation to the amount of computing resources, while minimizing its cost contribution. This is done by collecting and evaluating metrics to find under-/over-utilized resources.

1.3 Scope

Since applications and databases are run as resources hosted by a cloud service provider, the virtual machines tasked with executing code are separate from the scope of this context. Thus it is not possible in this research to examine the physical effects on the virtual machines and some details of the execution may be hidden. Monitoring statistics is more restrictive as a result. Data regarding the runtime of computing resources spent for respective databases are available, but it is not possible to necessarily control or observe how the distributed network of virtual machines calculates requests and variation that may follow as a result.

Within scope it is instead interesting to find information about how utilization and performance of resources are linked and be able to collect metrics that accurately represent an efficient resource depending on varying performance requirements. Variance as a result of the cloud environment are not generally tracked.

Performance optimization can be done in various ways with different levels of specificity for a given situation. For this paper, possible optimizations are restricted to compression and indexing. These optimizations were chosen as they are expected to have the most impactful results when managing databases with varying bottlenecked utilization metrics. Gathering metrics that can present database need for maintenance, or efficiency, can as such be much expanded upon outside the scope of this paper.

Chapter 2
Technical Background

Delivering fast, reliable and secure services require a robust IT infrastructure. To meet the growing expectations of the customers more processing power and storage capabilities are needed. Using an on-premise server solution means that computing resources need to be planned beforehand. Capacity planning when using cloud computing is slightly different, as it is easier and cheaper to request more processing resources. Instead there is a growing need to continuously analyze the current resource allocation and optimize usage. The cloud computing alternative becomes attractive due to more flexibility within a predefined environment, as opposed to building an infrastructure of resources on-premise [1].

Back in 1987 most users of database systems, at least for the time, preferred buying more resources instead of relying on technological improvements [2]. Most users relied on some rule of thumb for acceptable resource utilization levels and would allocate more resources to solve issues that did not align with the self-defined threshold. With resource allocation being cheaper and more available in cloud services this perspective is still prevalent today. Due to the ease of access to computing resources, and the trend of moving businesses to a cloud platform understanding utilization of computing resources is arguably more important now than ever before.

2.1 Working with Cloud Performance

As stated, one of the most troublesome aspects of on-premise planning for web applications are capacity planning [3]. For capacity planning the main concerns are what necessary computing power does the design and implementation of the web application require and what future workload is expected. Using a cloud hosted alternative, resources can be adapted to unexpected workloads much easier. Requesting more computational resources result in an increased cost, whether it is a pay-to-go or a subscription based payment model. This more flexible environment instead requires continuous analysis on what cost is necessary.

When evaluating performance of a cloud resource the most important aspect is that the quality of service (QoS) is sufficient for the end user. As such, monitoring availability of the system and response time as performance metrics is suggested as they are directly linked to the QoS demands. Response time consist of 4 parts; DNS time, connect time, server processing time, download time. In this paper

server processing time is the most interesting aspect, since the other parts are independent of the monitored data used to examine the resources.

2.2 Capacity Planning

Working with over and underutilization of resources, there exists studies regarding capacity planning that have been done for on-premise solutions. The most important part of capacity planning, and by extension a method to determine a reasonable amount of computing resources, is to have a well defined performance objective [2]. Certain features of the application should have a demanded valid performance level. That way, computing resources can be planned for this end goal. In this book, there is also a discussion surrounding capacity planning for composite measures of computing resources. While the composite metric is used to show the overall workload trend on a server it is difficult to establish an upper bound of capacity for the system using this metric. Bottleneck metrics are not easily recognizable as a result of observing the composite measure. It is, as they claim, necessary to consider specific resource utilization metrics to guarantee performance.

2.3 Database hardware functions

To understand how efficient a hosted database is it is important to understand how the database functions. The database works, similarly to any other computer, using CPU for arithmetic operations, memory for storing active data and disk storage which can contain more data at the cost of a longer time of retrieval. The Data I/O for a database refers to the cases where data is read or written from, or to, disk to memory. Since a database is meant for managing storage, it is often important to consider and sometimes minimize, slow operations such as managing disk storage. One way that can be used to circumvent reading from storage are using a buffer where active memory can be temporarily stored, just like a cache would in a computer. Log I/O writes to the transaction log for a given database as transactions occur in order to ensure the correctness of operations, should an error occur. These transaction logs tracks the changes that have been done to the allocated memory.

2.4 Execution plan

Execution plans describes the way a query is executed. The plan consists of several nodes making up a tree of blocking or nonblocking operators as depicted in figure 2.1 [4]. If an operator is blocking, it means that no output tuple can be created if not all the input has been read. A query can have multiple operators that are able to run in parallel. These concurrent parts are defined as pipelines. A pipeline start from the leaf operators until reaching a blocking operator. This ends the pipeline, if there are operators left then a new pipeline begins. Thus an execution plan can be seen as a tree of pipelines.

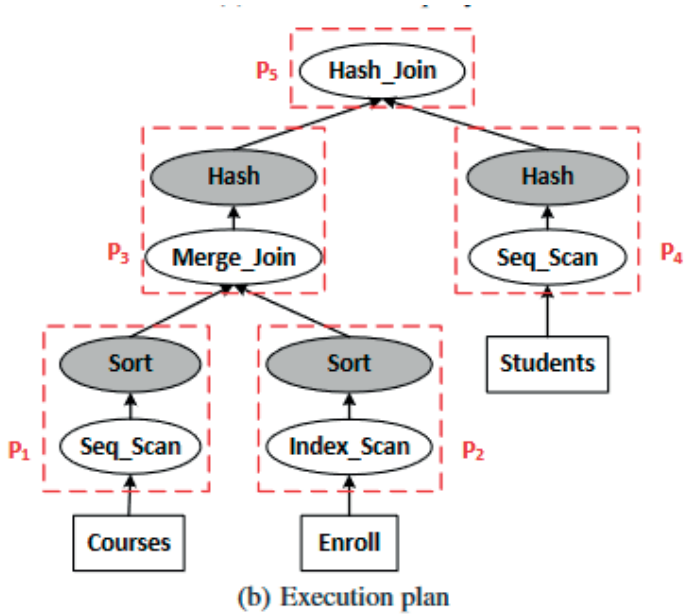


Figure 2.1: Example of Parallelization of execution plan

By examining this plan one can follow each individual step that is done during the query. This opens up the possibility for optimization, as bottlenecks can be seen more clearly. For example, if a pipeline performs a table scan, which tend to be slow, one can quickly see that it may be beneficial to implement an index. When a usable index is introduced the execution plan will automatically choose the new, seemingly optimal, plan converting a table scan to an index scan.

2.5 Compression

While the action of compressing information is a feature specific to relational database management. Configuring the database to efficiently store data is a core functionality for saving costs. It has been successfully tested as a potential improvement as a method that can greatly benefit, not only disk space, but also performance [5]. Many different compression algorithms are available, however there are two that are most interesting in this context, row and page compression.

2.5.1 Row compression

Row compression saves storage by reducing the metadata overhead, removing unused space such as NULL values and zeroes, and use the fewest possible bytes needed to represent some data types [6]. For example if an integer is between zero and 255, it is normally stored as the number of bytes used to represent an integer. However, since a value between 0 and 255 it can be stored using one byte. The variable length used by row compression stores that integer with one byte

and saves storage requirements as a result. However, if the integer is updated to a value that requires more than one byte of representing bits, the operation can become costly. The update operations needs to allocate more space to perform the update, which can have negative effects on the inherent storage techniques used for the database.

2.5.2 Page compression

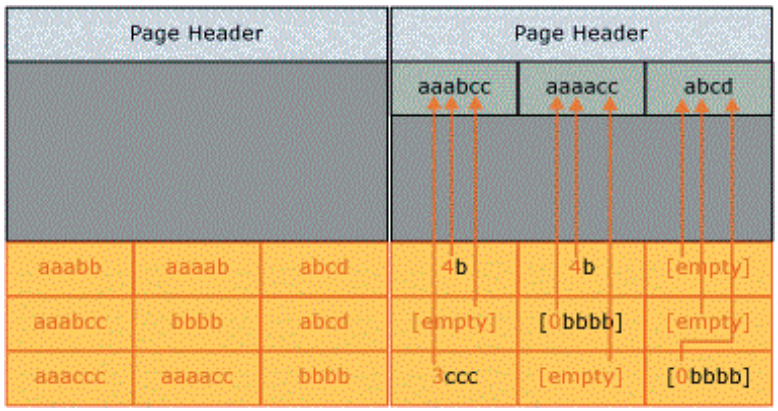


Figure 2.2: Prefix compression

Page compression is achieved by performing three operations in following order, row compression, prefix compression and dictionary compression. As shown by figure 2.2 [7], prefix compression is done by first finding a value for each column that can be used to reduce the values storage space in the column. Then for the prefix values of each column a row is created to represent it in the compression information structure. A reference is used to substitute all reoccurring prefix values in the columns. Should there exist a value in a row that doesn't match the chosen prefix value, then a partial match can still possibly be made.

Dictionary compression is performed after the prefix compression has finished. Figure 2.3 [7] shows how additional values are added to the page header dependant on what duplicate values are found after the prefix compression results. This compression checks the whole page for recurrent values and places them in the compression information area.

2.6 Indexing

Indexing is a very important concept to use to ensure that database operations are executing efficiently. By utilizing different indexing methods one can retrieve or search for data without checking the entire database table. There exists numerous methods for indexing such as non-cluster index and cluster index et. al [8]. Utilizing indexes effectively can lead to better query performance as fewer instructions are necessary to execute queries [9].

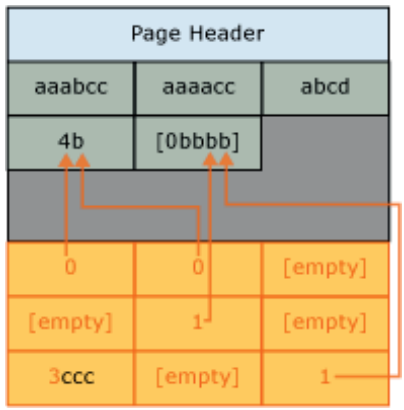


Figure 2.3: Dictionary compression

2.6.1 Clustered Index

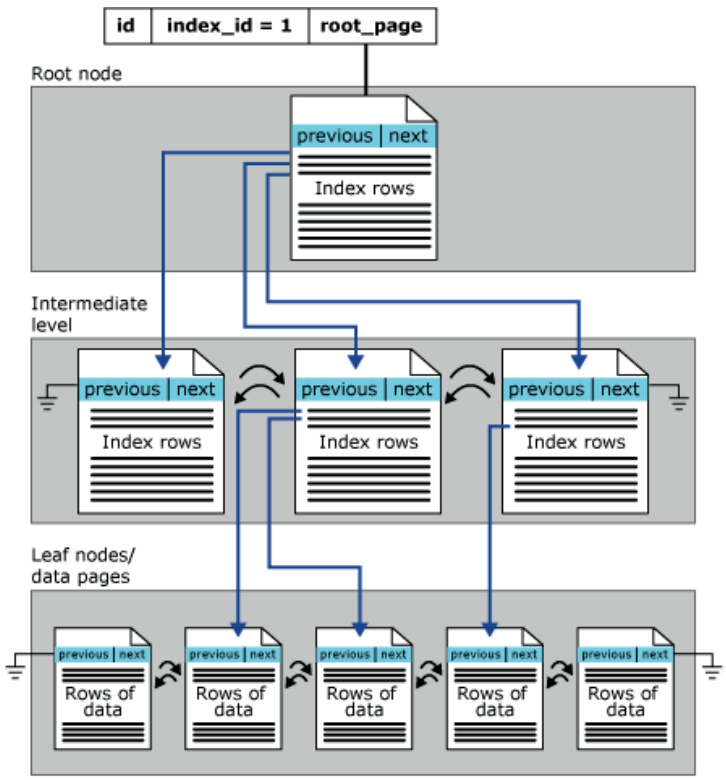


Figure 2.4: Clustered Index for a single partition

Clustered indexes uses the key values to sort the stored data. This is the only case when the data rows are sorted in a table [9]. As it is only possible to sort the

data rows in one way there can't exist more then one clustered index for a table. Should a table not contain a clustered index it is instead stored in a heap which is an unordered structure.

Figure 2.4 [10] depicts the implementation of clustered index in more detail. To perform aforementioned sorting of data, the index will create a B-tree. Key values used to sort data are stored in the root and intermediate levels of the B-tree along with a pointer to a lower level. The compound of key values and pointers are called index rows. The leaf nodes of the tree contains the rows of data and are, per partition of data, scanned via the sorted index rows. When ranges of keys are scanned, the previous/next pointers are used to sift through data.

2.6.2 Non-Clustered Index

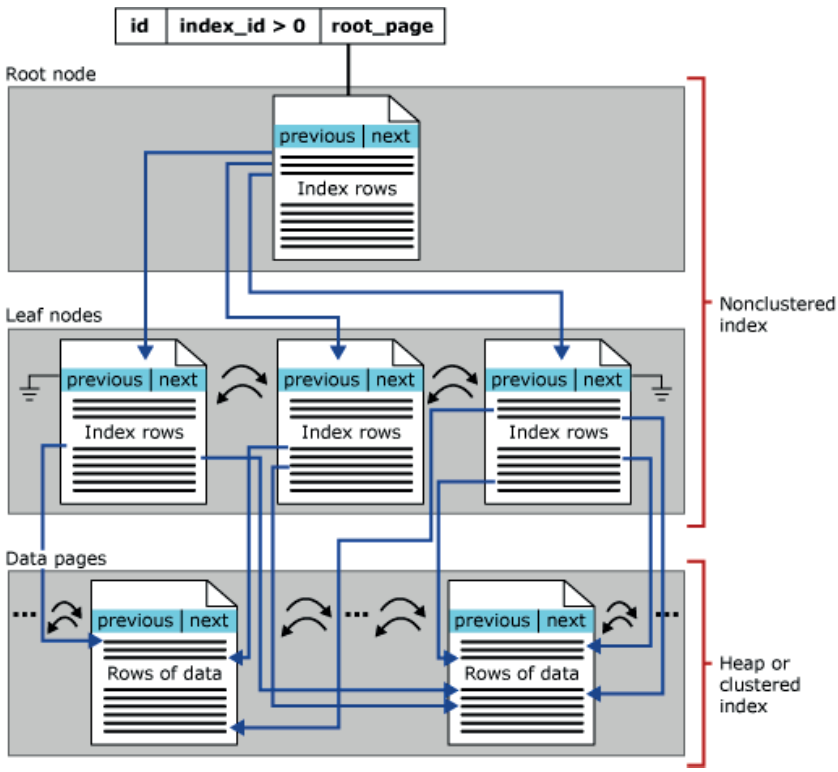


Figure 2.5: Non-clustered index for a single partition

The structure, as shown in figure 2.5 [11], is relatively similar to the clustered implementation as it also uses a B-tree structure. The two most notable differences to the clustered implementation are that the non-clustered index structure is a separate data structure from the rows of data and that the non-clustered implementation contains pointers to certain columns to the rows of data from the table. Since non-clustered indexes are data structures separate from the table data it is possible to have multiple non-clustered indexes but just one clustered index as a

result.

In non-clustered index, each index row is comprised of a key value and row locators used to locate specific data rows. A row locator is a pointer pointing from a non-clustered index key value to the corresponding data row. Depending on how the data pages are stored, heap or clustered table, the structure of the pointer varies. For heaps it points to the row, while for clustered tables the pointer is the clustered index key [9].

2.7 Important Metrics

There are many different ways to gauge the overall state of a database, depending on what the purpose is. In this context, the important factors are that bottlenecks can be identified, and that relative performance for a given utilization is justified. For this goal, three categories of metrics emerge; performance, utilization and database efficiency.

2.7.1 Performance

Performance metrics are most important for retaining QoS requirements as computing resources, and indirectly costs, will be lowered. The performance metrics we have chosen are:

- Server Processing Time
- Availability

These metrics are directly correlated to QoS requirements for the end user [3]. Regarding availability of our system, errors limiting the availability of the system are rare, but very important to note. The most common error that can occur is that a workload is too slow and causes a timeout error if computing resources are substandard. Server processing time will be the key performance metric as a result. While it may be possible to scale down computing resources at the cost of a slower server processing time, the most important aspect of this paper is to attempt to keep business critical functions at a reasonable server processing time despite having less resources.

2.7.2 Utilization

Utilization metrics are used to understand how much computing resources are currently used, the relative amount showing how much of the resources are being utilized. These metrics were suggested when managing energy costs for databases [12].

- CPU Usage
- Data IO Usage
- Log IO usage
- Disk storage

If the usage of any of these metrics are maximally used at any given time, it is likely that the performance will reach substandard levels for any workload. CPU, Data IO and Log IO are all key resources for performing operations on the database. Disk storage limits the amount of data that the database can store, and is also important for determining which tables of the database are costly and important to maintain.

2.7.3 Database Efficiency

Finally, database efficiency metrics will be used to verify that the current performance is necessary or if there are configurations that can be done to save costs. Since this paper is limited to considering optimizations regarding indexing and compression. Metrics describing database efficiency will be related to whether any of these operations may be useful.

Index Metrics

For considerations regarding where indexes can bring potential performance gain there are factors regarding the table and columns within that table as well as the query usage that affect the performance of an index [13][14].

- Column predicate.
- Future query usage.
- Column usage rate.
- Number of DML operations on the given column.
- Unique key constraint.
- Foreign unique key constraint.
- Nullable column.
- Distinct column values.
- Table size.

Column predicate is considered to be one of the more important metrics brought up. If columns are used in for example select or where statements it makes covering the column with an index a viable candidate. DML (Data Manipulation Language) operations refer to instructions such as inserts, updates and deletes. When deleting, or moving data, included in an index more overhead is necessary to exclude the data from the index and can lead to performance loss for these operations. This is also how the index can become defragmented, and less efficient overtime if no maintenance is done. Balancing selecting predicates against DML operations is paramount in how efficient a index is for the situation. Future query usage and column usage are important for understanding the potential performance gain by introducing a covering index that will see frequent use.

Unique key constraint, distinct column, foreign key constraint and non-nullable columns are all valuable because unique values in the column makes the index

creation more discriminating. That way the index can seek for partial results faster by having less data to examine, especially if those datapoints can be excluded from the search by using a lower density index. Foreign key constraints requires referential integrity, which indirectly requires unique values.

Finally, the table size is important both because of the potential gain when introducing index for a large table and the mistake of indexing a small table. For small enough tables it can be quicker to scan the entire table for results instead of the overhead introduced by finding the values via indexes and the relative performance increase for larger tables is dependant on how much data no longer needs to be scanned.

Compression metrics

To determine if a table is worth compressing it is important to determine how much of the data is compressible, what operations are performed on the tables, as well as the frequency of the operations [15][16]. Following metrics were investigated:

- In row data
- Row Overflow data
- Lob data
- Read percent
- Update percent

In row data, Row overflow data and LOB data are the three allocation units used to store data. The one that determines the potential for compression is the "in row data". This is the unit used when the size of a row is within 8000 bytes. The other two metrics represents the uncompressible portion of the table.

As for the operations related to compression the interesting ones are how often a scan or update operation is performed. The read percentage represents the amount of scan operations a table does while the update percentage is the amount of updates. With help of these metrics one can analyse the table and determine if it should be compressed with row/page compression or to leave it be. The characterization of usage over the database object is the most important aspect of determining compression. If the amount of reads are high and updates are low then page compression is recommended. If reads are middling, near 50% and updates are low then row compression is suggested. If updates are frequent or there are large amounts of uncompressible data, compression is probably not a suitable optimization.

Chapter 3
Implementation

3.1 Chosen Metrics to Analyze the System

These metrics were chosen from the related works as a means to monitor the state of the database for understanding the resource usage. These were divided into categories to easier explore and reason about them as seen in table 3.1. When these metrics were determined, a primary factor for inclusion was that the metrics were supported by more than one study. Furthermore, metrics regarding utilization and performance need to be sufficient to show some relation between performance relative to available resources. The metrics chosen should also be useful in the context of evaluating experiments.

Chosen Metrics	**Category**
Server Processing Time	Performance
Availability	Performance
CPU Usage	Utilization
Data IO Usage	Utilization
Log IO usage	Utilization
Disk storage	Utilization
Column predicate	Database Efficiency/Indexing
Future query usage	Database Efficiency/Indexing
Column usage rate	Database Efficiency/Indexing
Number of DML operations on the given column	Database Efficiency/Indexing
Unique key constraint	Database Efficiency/Indexing
Foreign unique key constraint	Database Efficiency/Indexing
Nullable column	Database Efficiency/Indexing
Distinct column values	Database Efficiency/Indexing
Table size	Database Efficiency/Indexing & Compression
Percentage of fragmentation in given table	Database Efficiency/Compression
High data volumes	Database Efficiency/Compression
Data growth/ Data decline	Database Efficiency/Compression
Number of update operations	Database Efficiency/Compression
Number of insert operations	Database Efficiency/Compression

Table 3.1: Table over chosen metrics and their categorization

Using the gathered metrics it should present possible optimizations for the database to alleviate resource usage as well as whether the suggested optimization will have a positive or negative impact. The chosen optimizations were compression

for managing Data IO and indexing for reducing necessary instructions for scan operations.

3.2 Using the Chosen Metrics on the Environment

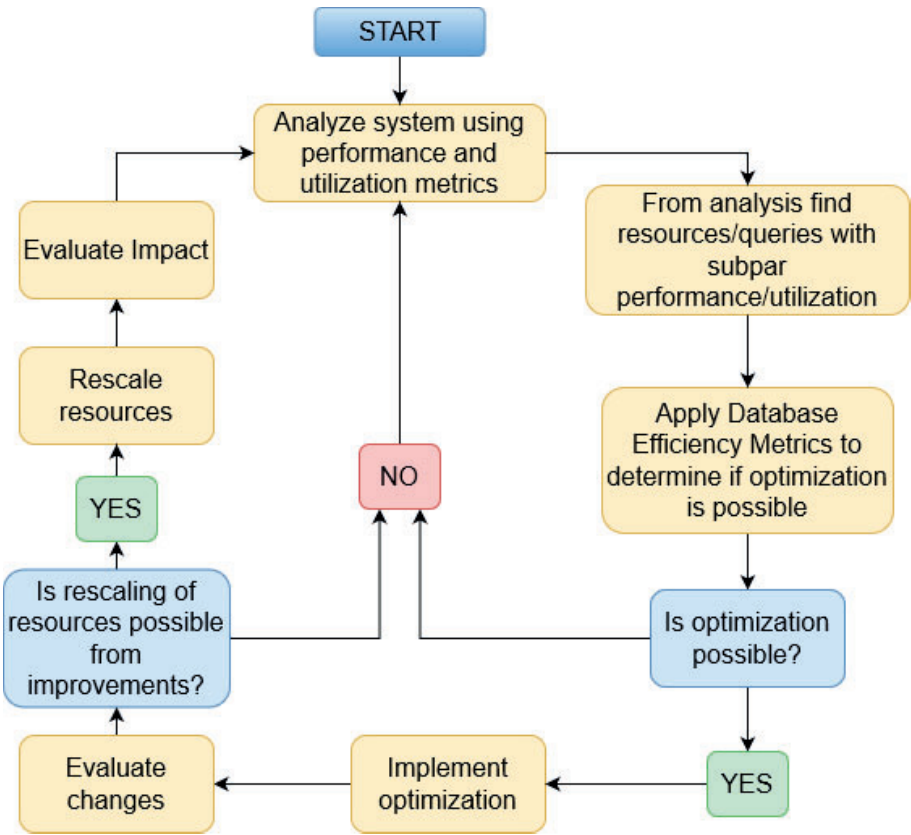


Figure 3.1: Workflow of Approach

In order to save costs using the chosen metrics the method shown in figure 3.1 was used. As a result of unknown impact of any given optimization or rescaling of resources there is a cyclic method of evaluating a change in relation to its previous state to see if costs can be reduced.

3.2.1 Analyzing System Utilization

The first step is to analyze the system performance in relation to its utilization. When high usage is detected it can reveal important performance detriments during certain time intervals. By examining the gathered metrics regarding utilization from the usage-spike one can identify bottlenecks and possibly characterize the workloads at the times of issue. This is done by querying the monitoring software

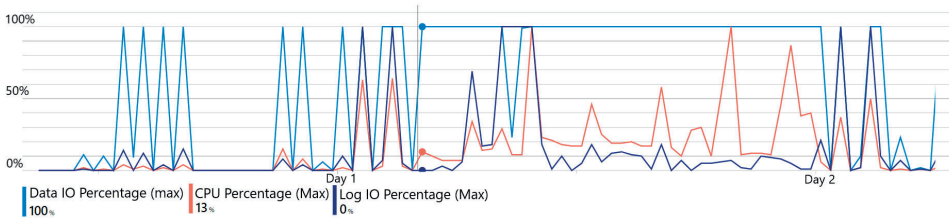


Figure 3.2: Graph of queried resource utilization of a database

for the cloud service platform. The utilization spikes as shown in figure 3.2 are matched to the active queries during this time interval. Each query represents a certain percentage of resources used. A simple tool, see figure 3.3, was developed to match the cases where maximum utilization were found to the active queries and their relative usage at the given time. This supports identifying problematic workloads and the reason why they behave a certain way.

3.2.2 Database Efficiency Metrics

From the analysis resources/queries with subpar performance are found. With this we apply our database efficiency metrics from table 3.1 to determine if optimization is possible for found resource. Should the metrics indicate that the optimization would bring improvements, then it is implemented, if not then we repeat our analysis to find a new resource.

After introducing a optimization it has to be evaluated to confirm the validity of the chosen metrics and the analysis. This is done by inspecting the change in performance and utilization metrics. These metrics are collected by utilizing various SQL scripts to extract raw data directly from the Dynamic Management View (DMV) as well as performing calculations on the relative change for the data. Data from the DMV is presented with an aggregate interval of 1 hour in this environment. This makes the exact values of queries ran multiple times within an aggregate interval unidentifiable. As a result, the average cost is collected within each aggregate interval of an execution plan, which may not be exact but should still give a good approximate cost evaluation for said query.

This workflow of gathering metrics is described partly in figure 3.3. Depending on the results from the evaluation we determine if rescaling the resources is feasible.

3.2.3 Rescaling Resources

After an optimization has shown improvements, down-scaling a resource is possible if the reduction in performance is expected to be at least similar to the database state before optimization. Critical queries, those that were either related to high utilization or business critical queries that affect the customer facing features are most important. Lowering the amount of processing power is shown in figure 3.4, where DTU is the nomenclature used in the cloud service for the composite measure of CPU, Data IO and Log IO.

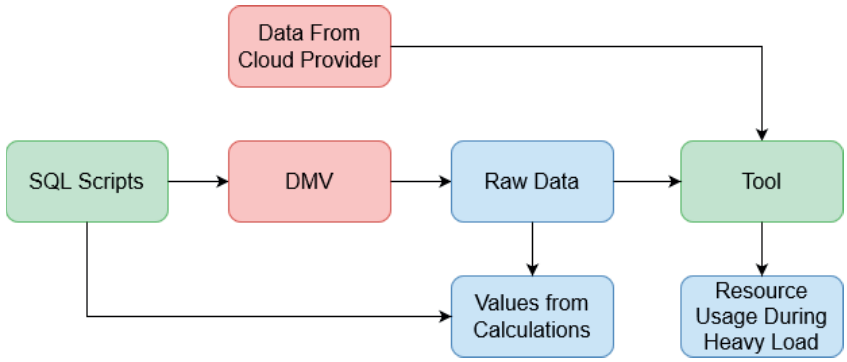


Figure 3.3: Workflow for gathering metrics

Figure 3.4: Rescaling resources is done by simply changing the value on a slider

After this is done, metrics regarding utilization and performance will be gathered over time to ensure that QoS is still being met. If performance is sub-standard it is necessary to analyze the database again.

3.3 Working Environment

For the database architecture it is important to understand how data is being propagated to the server, since this influences the design. Data is obtained from an external API gathering large amounts of data as seen in figure 3.5. The data is then sent to the system by scheduling events daily, weekly or monthly through a function represented by the diamond symbol in the illustration. This function is ran every hour and checks if a scheduled job should be started. Should the job fail or not started according to it's scheduled time the event will be rescheduled until it succeeds.

There are two different kinds of jobs that function does. The first is simply to extract, transform and store the data in SQL tables. The second is to perform calculations on data which are then stored to be used by the downstream systems.

Jobs that are done either daily, weekly or monthly are all scheduled to occur in off hours, as they tend to be rather demanding. New data should propagate to eventual tables for the databases and old data will be inserted in an archive. When these large amounts of data are updated indexes are dropped and later rebuilt when the data has finished updating.

The databases within the server are separated into their categorical usage. There is one database that is mainly used for reports over a given month. An-

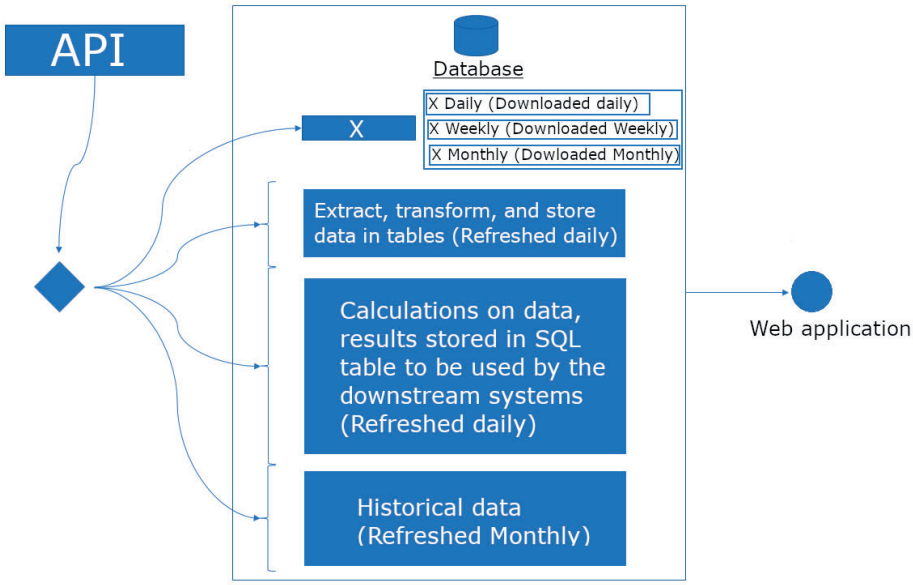


Figure 3.5: Overview of working environment

other database in the environment, collects various data which would be slow to find if it where to be collected impromptu when requested by the web application. Information collected by this database is slow to get as a result of needing to combine data from multiple different tables and eventual calculations on the gathered data. The data in this case is stored as read-only after it has been populated during off hours as a means to quickly lookup this information. The third and final database is used for most basic demands of the web application. Whereas the read-only database contains more complicated information, this database contains data that is easy to manipulate and present using the application.

This database design is reoccurring for multiple different server instances. These instances are a development environment, test environment, production environment and a simulated production environment. The different instances have different allocations of data. Certain functions are run for the given environment, such as deployment and smoke tests on the databases, where some functions are excluded/included depending on instance.

When attempting to save costs, these are the databases and different environments that can be examined. For the following results, the databases that will be investigated are the databases for the simple application features in both the development and simulated production environments.

3.4 Testing Environment to Evaluate Changes

The environment that was used in this context has, as mentioned, a related web application that the databases are responsible for. Certain functions within the web applications are considered business critical as a result of directly correlating

to the QoS when using the web application as a customer. When testing business critical queries they are triggered using the web application. Metrics are gathered and analyzed after several tests of the queries, during times where unexpected workloads should not interfere with the comparisons between different states of the database.

However, there are not only business critical queries considered when analyzing usage for the database. There are multiple scheduled queries that run concurrently during the day as well as daily, weekly and monthly batches of queries that are triggered during off business hours. These queries cannot be forced to run during particular times, and will be collected over longer periods of time to be less affected by abnormal behaviour. These times were data was collected can vary between a month, or a week, depending on how often the query is scheduled to run.

Chapter 4
Results & Discussion

4.1 Examining Problematic Databases

When examining resources it is particularly important to look for those that have utilization metrics at near 100% during a common workday, as these are required to have a certain level of stability in performance. If resource usage tend to be too high, any further usage of the application may be constrained by resources as a result.

The database for application features resource usage is shown in figure 4.1. The resource usage is the statistics showcasing the most expensive amongst CPU, Data IO and Log IO during runtime over a week.

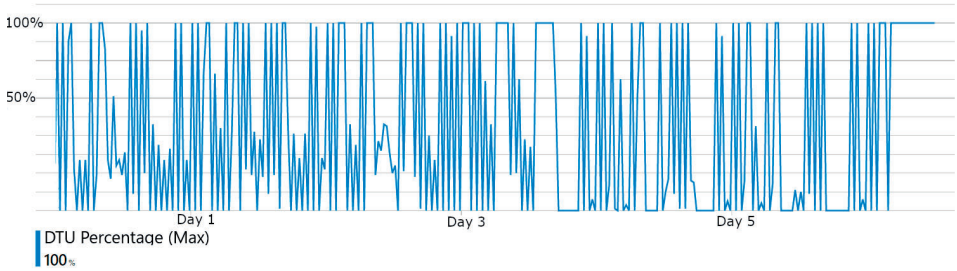
In figure 4.1a and 4.1b the graph is a compound statistic of CPU, Data IO and Log IO and shows the one that has the highest utilization at any given time. From this data it seems to be a significantly over-utilized resource. This is in the developmental environment for the database. These factors means this particular database is an interesting one to test metrics both when a resource is overutilized and if using the suggested database efficiency metrics can save costs. At the current state, this database is considered purposefully under-provisioned as the performance only affects developers, and it is cheaper. While results from optimizing this database may not necessarily impact the cost as drastically as a more expensive production environment, results from experimenting with this database should still be applicable to other situations.

4.2 Analyzing the Database in Detail

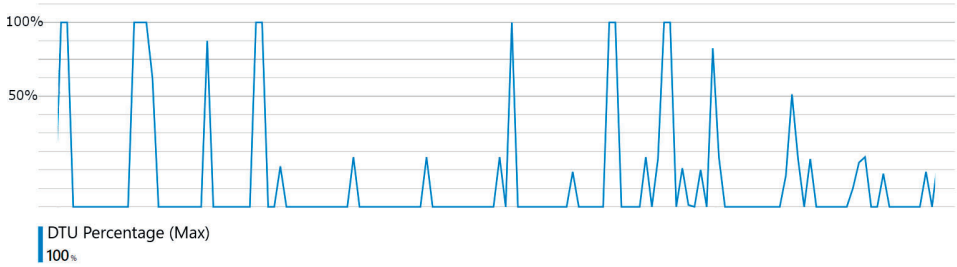
From the utilization chart of this database, as seen in figure 4.2, it seems that Data IO tends to be the most prevalent utilization constraint. During the extended 100% spike occurring on Sunday, the database's most important demand is that queries do not time out.

4.2.1 The Most Important Queries

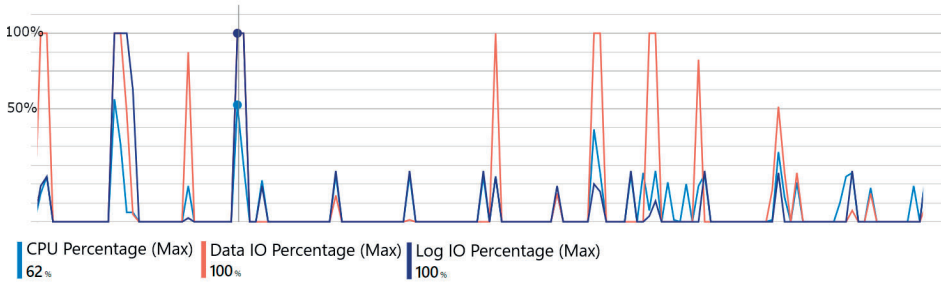
There are two kinds of important queries in this case. The queries that demand high resources at a specific time and those that run frequently and put a consistent average workload on the server. Both of these kind of important queries constrain



(a) Resource usage over a week



(b) Resource usage over a day



(c) Resources for the same day split into CPU, Data IO and Log IO

Figure 4.1: Utilization of database for application facing features

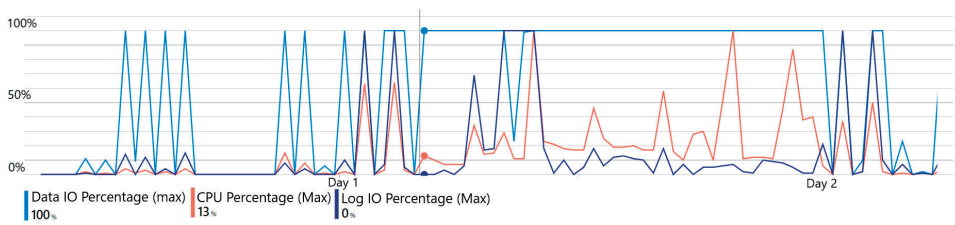


Figure 4.2: Resource usage before compression

how the server eventually can be rescaled. Should the most expensive one-time queries be left under-provisioned they may time out and cause inconsistent behaviour as a result of scheduled reruns at worse times. If a query that places a higher average resource cost on the server throughout the day it is important to consider if application functions that may run concurrently are responsive enough to remain at a stable performance.

Expensive Queries

Since Data IO is the most constrained utilization metric, the most expensive Data IO queries were found.

1. Query 4822, a SQL defined stored procedure to collect statistics regarding a certain table. It ran about 70-80 times per week.
2. Query 5048, this query merges old data from two tables to the archive. It is run every weekend.
3. Query 22525/21826/21805, are sql stored procedures to update statistics on other tables.
4. Query 4713, a bulk insert statement that populates a particular table with between 20 to 5000 rows sometimes more often than once per hour for any given day.

The stored procedures having such a high impact on the server utilization may be a result of the database being low provisioned. The only optimization that can be done to these queries are to set them to collect statistics more infrequently. However, it is still possible to consider query 5048 and query 4713 for improvement using the gathered database efficiency metrics. These two queries span different kinds of resource demands on the server. Where optimizing query 5048 may make the off-business hours more effective and query 4713 would put less pressure on the system throughout all usage, effectively leading to the database having more resources available for any additional workload.

Business Critical Functions

For this context, business critical functions are client-facing web application functions. The most expensive functions make multiple concurrent calls to the database to present a result. In order to meet QoS all client-facing functions need to finish execution within a certain server processing time consistently. In order to explore this, the performance objective set was that business critical functions should retain the same server processing time at lower resource provisioning over multiple calls.

Testing all client side functions was not easily done. The most expensive functions were tested until the determined performance objective failed. This method makes it difficult to show that everything meets the required QoS after changes, should no business critical function observed offend the QoS. Then there would need to be a guarantee that all functions would perform sufficiently well under any reasonable concurrent workload. If the most expensive functions fail

QoS it is then easy to disprove that the current resource allocation meets the necessary performance objective. In short, there are many parameters that need to be taken into consideration when proving that business critical functions follow the set performance objective. Instead, if exceptions to the QoS is found it is simple to show that the current performance is not acceptable and in need of changes.

4.2.2 Query Duration can be Deceiving

As queries were examined in relation to the worst performing in regards to the utilization demand on the database, it showed that only examining the duration of a query may be misleading.

Figure 4.3 reveals that the amount of utilization required by concurrent queries has an effect on the duration of each query. The figure shows utilization metrics and server processing time, divided into queries at a particular point in time where overall utilization was 100% for some metric, relative to the percentual utilization used for the given statistic. Even if resource usage is low the duration is higher then expected as a result of the constrained resources.

Another interesting finding is that, as seen in figure 4.4, the Data IO resource usage is high for the purple query while its duration is comparatively low and the other query uses a similar amount of Data IO but duration is much longer. This insinuates that the resulting duration in relation to the utilization metrics can be misleading as they are either not directly linked or inexplicably vary depending on concurrent server workloads.

As seen in the results one cannot blindly use the query duration to gauge the performance and efficiency of the database. One also has to take into account that the duration is or can be inflated by other queries running concurrently. This case is exemplified by the results seen in figure 4.3 if considering the duration of the orange query. Here it can be seen that a query that use few resources still have a noticeably high duration. Thus it is possible to be deceived by the duration of the query resulting in wasted time trying to optimize resource usage for this particular, leading to no significant resource improvement. Instead if the orange query is always expected to be run concurrently with certain more demanding queries it might be more beneficial to optimize those in order to have more stable performance for the orange query.

Even when there is, as few as, two concurrent queries parallelization can cause problems. As seen in figure 4.4 there are only two queries causing the spike in resource usage. If query duration is the metric used to identify issues it would clearly seem to be only one problematic query in this instance of max resource utilization. In this case, that's not entirely true. Even though the purple query has a relative low duration compared to the other it still uses up just as much IO resources, which also bottleneck the application. Therefore there may exist a possibility to miss the short and resource demanding query, should only duration be taken into account. By limiting the scope of the chosen metrics to only observing problematic queries in relation to performance of that query in certain instances may lead to missing vital information. It is potentially deceiving to consider only the query itself, and not the effect that concurrent workloads result in.

This can make server processing time a performance metric that is somewhat deceiving as one has to consider the surrounding queries when trying to optimize databases.

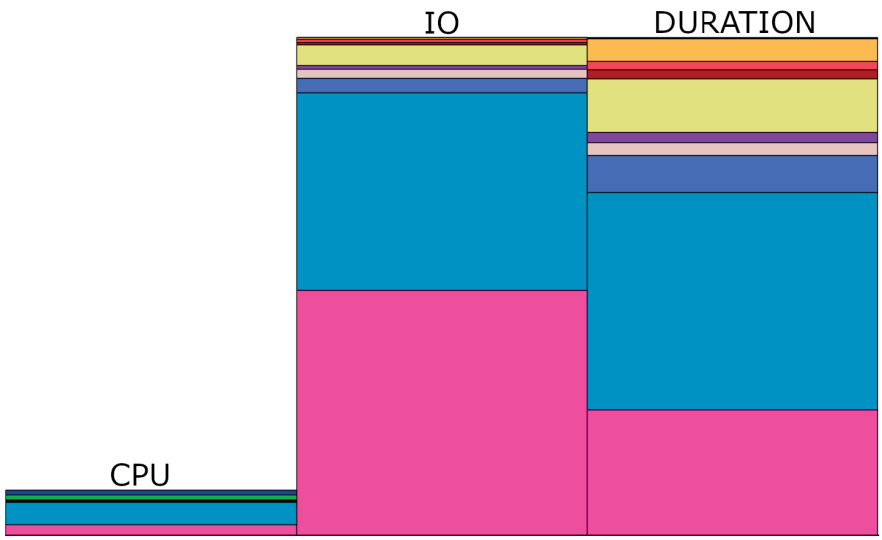


Figure 4.3: Bars from left to right: CPU, Data IO, Duration

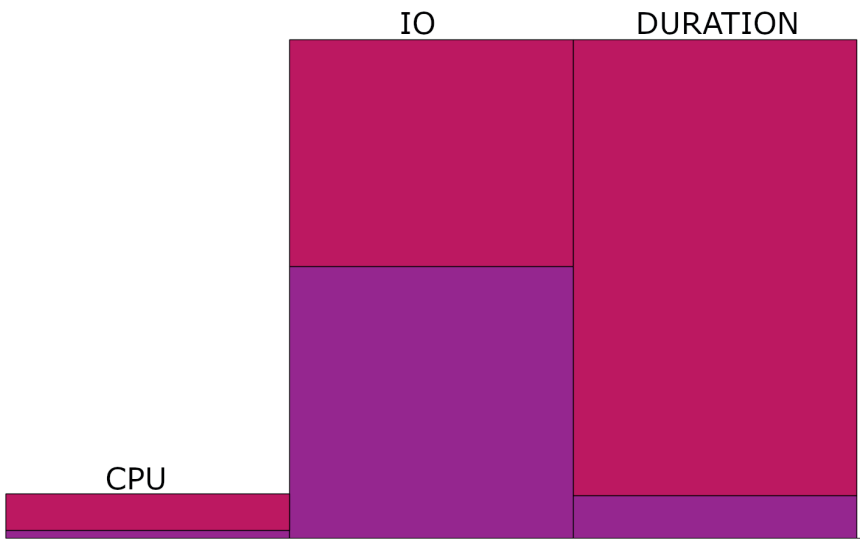


Figure 4.4: Bars from left to right: CPU, Data IO, Duration

4.2.3 Applying Database Efficiency Metrics

At this time metrics were collected suggesting if compression was a reasonable improvement to make. As can be seen in table 4.1, both tables consist of solely compressible data, contain relatively large volumes of data and the usage of these tables make them good candidates for compression. Curiously the update percentage is zero. This is because the tables are repopulated instead of updated. Large scale deletes and inserts occur as part of the nightly batchwork instead of updating rows to current values. Insert operations contain some extra overhead, especially for page compressed tables but are not nearly as big of an issue as updates can become. Updates are far more cumbersome as storage may need to reallocated during the operation.

Table	In row data	Lob data	Read percent	Update percent	Size (MB)
Archive table Q5048	4 238 976	0	100	0	33 117
Q4713 table	401 945	0	57.39	0	3140

Table 4.1: Compression Metrics over tables included in problematic queries

Considering index as a solution requires firstly to consider the column predicates. Since query 4713 is a bulk insert statement, introducing an index would only make the query slower. Insert is a DML operation and would only require more overhead if more indexes where involved. For query 5048, it has been stated as a merge statement between the archive and two tables joined together using an outer join. The cases that should be joined are where there was not a matching case using multiple parameters and the date column of the row is 2 months earlier than now. What this means functionally, is that the cases that do not have active issues, as defined in one of the tables, and are 2 months old are moved to the archive. Considering an index optimization for this query is problematic, the tables joined already use an index in that part of the pipeline. There is a table scan used for the archive table where data is merged to, but in this part of the pipeline there is no predicate to discriminate search results as it stands. As such indexing does not seem to be useful to optimize this query.

4.3 Optimizing Utilization of the Database

4.3.1 Page Compression Results for Query 5048

After the compression, the server processing time is now 67% faster for query 5048. The CPU time is 37% less and the number of physical reads are 53% fewer. Essentially the performance is faster, and the necessary utilization resources is notably less for both CPU and Data IO. The expectation was that the CPU utilization would be slightly higher as a trade off for lower Data IO usage, but it seems that was not the case. The number of physical reads are simply fewer after compression in this case, which likely made the CPU time lower as well. The number of necessary logical reads are not fewer, especially since the execution plan remains unchanged. Once the table was compressed, there was more available memory in

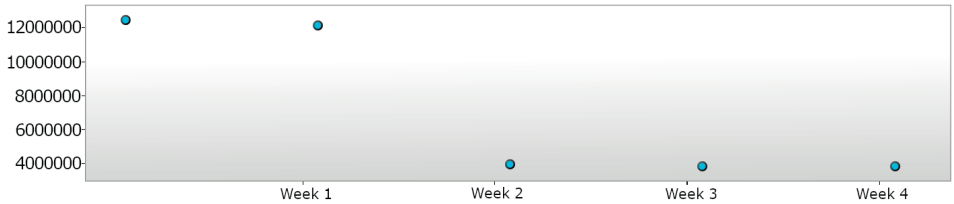


Figure 4.5: Difference in physical reads for query 5048 over a month

the cache for the query to run more efficiently since data that is compressed on the disk, remain compressed in memory [16]. As such it was possible to see a strict improvement for the query using compression.

Considering the utilization graph after compression, weekend batch jobs are much less demanding. There does not seem to be higher CPU usage outside of the observed query either.

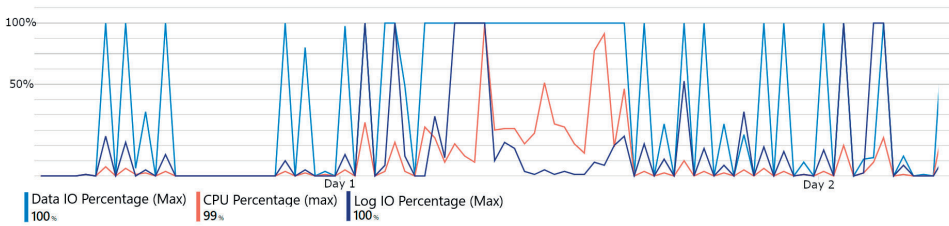


Figure 4.6: Resource usage after first compression

4.3.2 Row Compression Results for Query 4713

Comparing the duration before and after row compression was performed on the related table, query 4713 is not noticeably different when only looking at figure 4.7. The reason is that the number of rows affected by the bulk insert statement varies whenever run and this variation is aggregated over a longer time in the graph. There is also a lack of precision in the figure due to canceled (squares) and aborted (small triangles) runs. The reason for the canceled runs during this time is unknown, but it is not a reoccurring phenomena and are considered statistical outliers. The aborted runs on the other hand consistently occur every sunday, independant of the compression. As we are unsure of the variation that occur statistically for these aborted runs, and knowing they are not necessarily correlated to the optimization, both canceled and aborted runs will be excluded from the statistics for this query.

When observing the computation relative to affected number of rows it is more apparent what occurred after optimization. Naturally, comparing executions equally for drastically different workloads, or rather affected rows, does not yield a reliable result. As such, a metric that should be included for performance evaluation is rows affected. For the data collected after the optimization the aver-

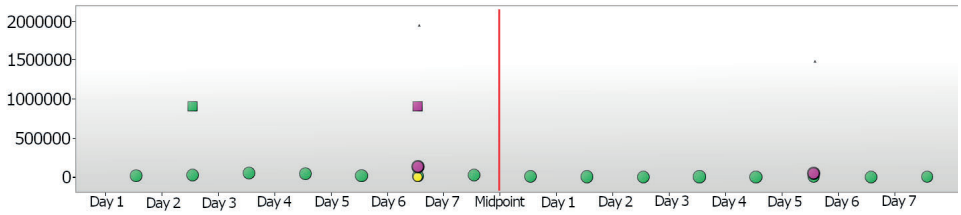


Figure 4.7: Difference in physical reads for query 4713, 1 week before and after row compression

age number of rows is 6% higher, which is only slightly higher and should not be expected to cause different results due to overutilization of larger workloads. Then there is a performance increase, as server processing time is down by by 63% per row. Number of physical reads has decreased by 80% and CPU time 68% lower. Once again the number of required IO operations is significantly lower despite the execution plan being unchanged.

The utilization graph after row compression shows much smaller data IO usage outside of the large weekly batchjob.

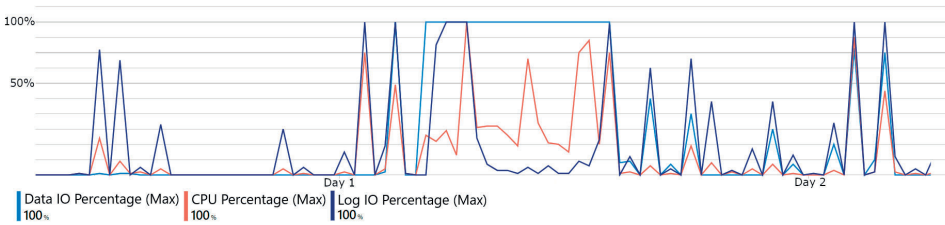


Figure 4.8: Resource usage after second compression

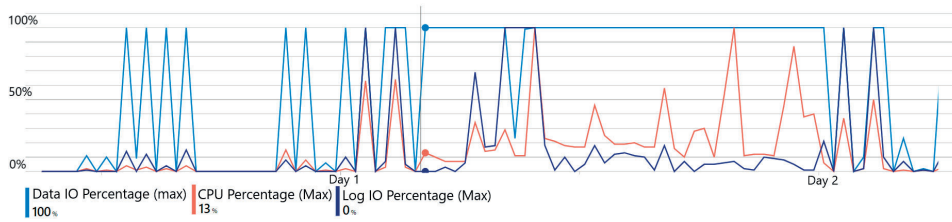
4.4 Rescaling Resources

Ultimately, changes that have been done previously aim to reach a point where scaling down computing resources can be done without negatively impacting performance. For this article the possible levels of resources are limited to be 40% of current computational resources, double resources or to stay at the current level.

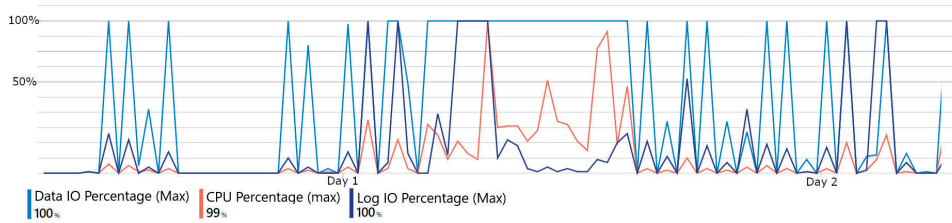
4.4.1 Comparing the Database after Improvements

All following utilization graphs where taken between 9:00 AM Sunday to 6:00 AM Monday with 30 min granularity. These are mostly scheduled times and show consistent behaviour any week. These graphs are mainly used for presentation, and the workload over the entire course of the database is considered when analyzing the database state.

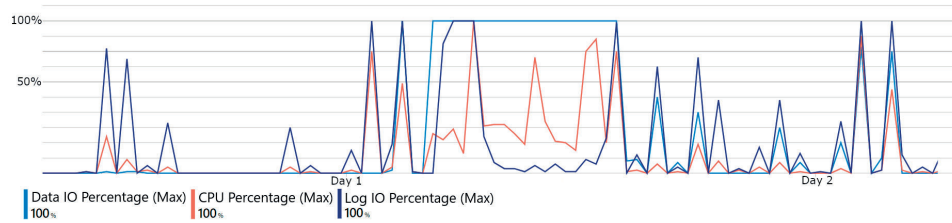
In figure 4.9 there are improvements in utilized computing resources when comparing the database without optimizations to having compressed two tables.



(a) No Compression



(b) Page Compression on archive

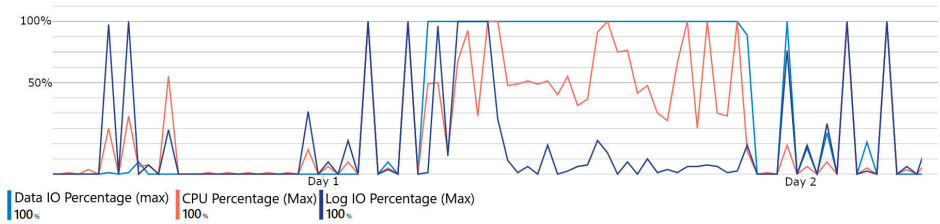


(c) Row Compression

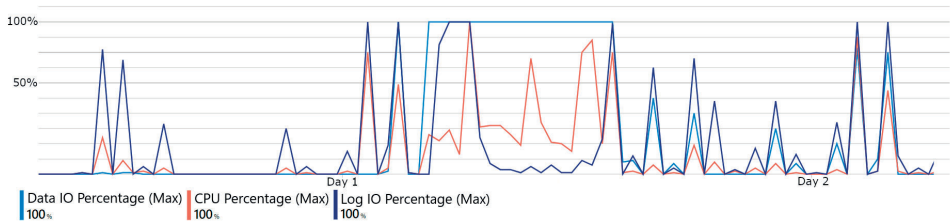
Figure 4.9: Resource usage at default resources

The question is whether this is enough to scale down resources to 40% of current capacity. Business critical functions, those that may occur as an unexpected workload, may be affected at this large of a difference. The application functions tend to not relate to high levels of resource utilization and do not seem to need direct optimization but rather stable average usage for active hours. What effects of having lower resource provisioning for these functions are as such unknown, as well as the overall efficiency of scheduled jobs.

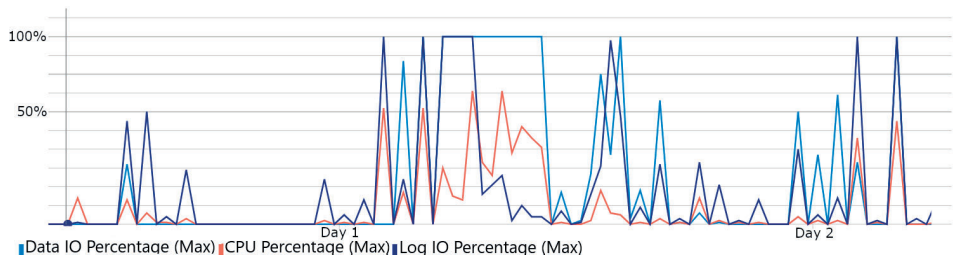
4.4.2 Utilization Graphs for the Different Resource Levels



(a) 40% computing resources



(b) Default resource allocation



(c) Double resource allocation

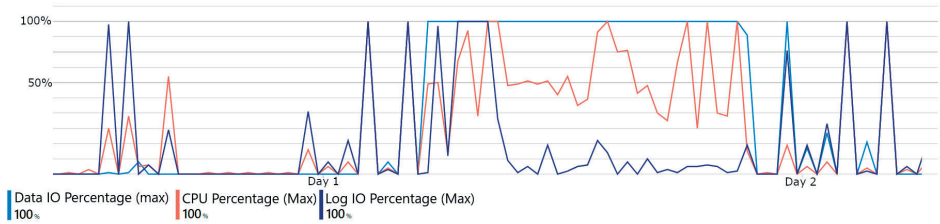
Figure 4.10: Utilization Graphs for the Different Resource Levels

As seen for the variations within figure 4.10 the first section of spikes before the end of the first day show a fairly expected outcome. If a workload takes 80% at the default resource provisioning it should become relatively smaller as additional resources are introduced. The resource allocation is described to act linearly in documentation for the platform. Theoretically, if a query costs 80% and double resources are provided it should now require 40%. Referring to the results of these graphs, at the relative demand in the first day of the graphs for log IO

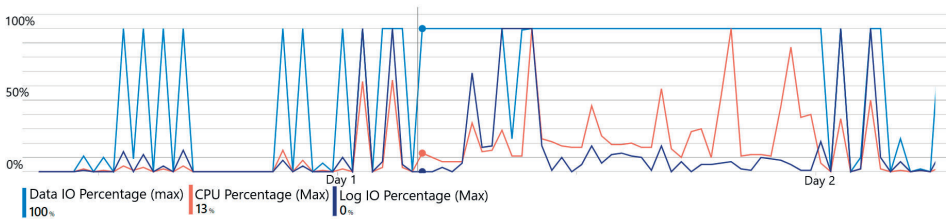
goes from around 80% at default resources to nearly 60% at double resource. The relationship for resource utilization for different provisioning does not act linearly when tested.

The most notable difference is the time it takes to execute the weekly jobs at the end of the week. Unsurprisingly, when a time interval of max utilization is encountered, provisioning more resources will make that period shorter. Doubling the amount of resources is not enough to lower the relative utilization at the time below 100%. In other words, if an unexpected workload would be applied at this time it would be unlikely to finish in a timely manner. Since this is off hours the important thing here is not that unexpected workloads can be managed but rather that the batch of queries can be successfully performed.

Jobs performed after the large batch of end-of-the-week queries do not see significant changes in the utilization graph regardless of resource provisioning. This may or may not properly indicate the performance at this time and will be analyzed further.



(a) 40% resource allocation



(b) Database before any optimization

Figure 4.11: Resource usage comparing 40% scaling and the database without optimization

When comparing the utilization graphs between figure 4.11a and figure 4.11b it seems that the 40% resource provisioning performs better than before any compression or scaling. The jobs at 12 PM are constrained by log IO and data IO respectively, but the weekly batch job is faster although the relative CPU usage is higher. If only the utilization graphs were compared, as was the strategy most often used [2] and still used previously in this context, it would be reasonable to conclude that performance would be expected to be at least similar.

4.4.3 Comparing Query Performance at Different Resource Levels

To verify that different resource levels still perform well enough, queries that were previously targeted for optimization and business critical functions will be tested. If these perform at least similarly to the unoptimized database, the downscaling of resources was successful. It is also important to note that queries observed in this case are representative of the database usage and does not cover all performance changes. As stated for business critical queries, this strategy is limited in what can be concluded if QoS seems to be fine as in theory every query related to the performance objective needs to be examined under various circumstances. If QoS is not reached for a lower resource provisioning for any query related to the performance objective of ensuring business critical queries run within similar time it can immediately be stated that the current state of the database is unacceptable.

Data provided in the following tables are metrics relative to the default resource allocation. This means that if duration at a certain resource level is higher than one it is relatively higher compared to the default resource provisioning.

Business Critical Query

Resources	Duration	CPU	Logical IO	Wait time
40%	1.986	1.344	1.049	8.351
200%	0.416	0.888	1.010	0.281
200% / 40%	0.336	0.747	1.008	0.051

Table 4.2: Performance of Business critical query when scaling

Immediately we found that the relation between server processing time and utilization metrics were not sufficient to explain the changes in execution for different resource allocation. Then as more research was done it became evident that considerable wait time for resources is a sign of under provisioning [17]. Wait time is a performance metric describing what time was spent to await resources to perform CPU operations and it is evident from table 4.2 that this query is severely overutilized at 40% of the resources. The duration has doubled which seems to mainly be a result of the wait time that increased eightfold. CPU duration also increased by 30%.

Something to keep in mind is that this query is called concurrently with many other queries. This affects the performance of the query especially when lowering the resources. The reason for this is that by lowering the resources we also limit available resources for concurrent queries as seen by the wait time. So instead of seeing a linear decrease in performance, a much worse result is observed.

When increasing the resources we see a improvement in duration of 60%. This increase is mainly from the improved wait time which improved by 72%. The CPU time performed 12% better which only slightly impacts the total duration time.

When comparing 200% to 40% we see a drastic improvement of 95% in wait time. Considering that the logical operations stay relatively similar, the execution has not changed drastically but rather the availability of resources. The other metrics show minor improvements when comparing 200% to default resources.

From the results in table 4.2 it seems that wait time is an important metric when trying to measure the performance of the database in relation to the utilization of resources. The business critical query is no longer compliant with QoS after downscaling, since the server processing time doubled.

Query 4713 - Bulk Insert

Resources	Duration	CPU time	Physical IO	Wait time
40%	1,147	1,143	1,183	1,323
200%	0,019	0,690	0,636	0,075
200% / 40%	0,017	0,607	0,540	0,058

Table 4.3: Resource scaling of query 4713

Not evident from table 4.3, but still important to note is that each weekly batch there is an aborted query 4713 that was not present at double resources. The time observing this phenomenon was limited to one occurrence and will therefore not be considered a result although it may mean improved availability for this query at higher resource provisioning. Since it is not certain whether higher resources solve the availability issue, the numbers in table 4.3 excludes failed queries.

The average server duration for this query is approximately 15% slower. Considering that the compression saved 63% from the original state for the database this level of performance is still significantly improved after scaling down resources. One of the most resource consuming queries on the database is still performing relatively well, given the new circumstances. It could be a result of the fact that this query does not tend to use maximal resources at a specific time but rather puts a continuous pressure on the database when running other, more time sensitive queries.

Comparing the results from 200% resource usage however, implies that the current resource allocation was under-provisioning the database. The average duration of the query decreased by 98% and while CPU time is not nearly as drastically reduced it is clear that the time spent waiting on CPU resources plays an important role. The physical IO is also significantly smaller, which can be a result of more effective cache usage at higher resource provisioning. Seeing how the time spent using the CPU is strictly lower for the same operations. This could mean that the actual execution of queries may be more efficient when more resources are available and possibly vice versa.

The comparison between 200% resources to 40% show just a slight difference to 200% compared to the default resources. While the query at 40% performs well relative to the default state, this query is rather under-provisioned when comparing to what it could perform at more resources. Within the quality of service demands, the query is completely fine.

Query 5048 - Merging Archive Data

An important distinction to make regarding these results is that the rescaled testing periods were a week long. As a result this particular query is only ran once,

resources	duration	CPU	physical IO	logical IO	wait time
40%	1.711	1.342	1.016	1.021	1.243
200%	0,553	1,061	1,101	1,074	0,362
200% / 40%	0,343	0,831	1,142	1,091	0,271

Table 4.4: Resource scaling of query 5048

as part of the weekly batch, for all resource levels excluding default. There is a certain level of statistical uncertainty regarding these results.

After compression this query was 67% faster. Once scaled down it takes 71 % longer relative to the new server processing time which is still a net positive. Other timing statistics such as CPU time and wait time increase in line with this change.

Comparing the results for 200% resources the duration is significantly lower at nearly 45% less server processing time. This time saved seems to come from the lower wait time. The CPU time and number of reads from both memory and disk are relatively unchanged. The execution itself has not changed outside just waiting. Where Query 4713 could use fewer reads from disk, because of the larger memory available at the time of execution, this query is still constrained during a time of maximum utilization usage, see figure 4.10c. Because the query cost, relative to available resources, is high the system cannot minimize disk usage and further rely on the cache.

At 40% resources the CPU time is higher while physical and logical operations remain relatively stable. This is something that should be explored further using a profiler to understand how the cpu time is slower with less available resources and otherwise similar instructions. Unfortunately, there is no profiling tools available for monitoring cloud databases at the time for testing these values more specifically. This result could also be a result of statistical uncertainty and the effect of concurrent queries that have not been considered during the particular query instance.

4.4.4 Wait Statistics

From our results of rescaling resources came the important realization that the degree of parallelization can start to cause issues once under-provisioned. Even as the utilization graphs and top consuming queries do not show any particular problems, we find that the business critical functions performance have significantly worsened. As resource utilization by itself doesn't show the resource demand of a workload, wait statistics should be considered an important metric to investigate as a complement to resource utilization [17].

A possible issue is that the business critical function is a collection of concurrent queries, as opposed to the expensive queries that may be run concurrently with other queries. When the database is under-provisioned and the business critical function is invoked as multiple queries the parallelization problems are evident due to the amount of waiting required.

Further exploring the relationship between under-provisioned resources and wait time can be useful, especially in cases where concurrent queries are ran as a

result of the application.

4.4.5 More Resources Does not Guarantee Performance

Something observed is that a query can regress in performance as a result of the execution plan. The execution plan is chosen based on the cost estimate [18]. This estimate can vary in accuracy depending on the underlying statistics. Should they be outdated it can lead to sub-optimal plans being used, resulting in performance degradation of the query.

An instance of this happened to one of queries related to expensive business-critical function. As resource provisioning was increased, performance improvement was expected, but the opposite effect was observed for one of the concurrent queries as shown in table 4.5. As stated in the official documentation for this particular environment, the execution plan is chosen based on the expected load placed on the server [19].

scaling of resources	duration diff	CPU diff	physical IO diff	logical IO diff	wait time diff
200% (new execution plan)	1.520	2.710	25.848	0.051	0.568
200% (old execution plan)	0.544	0.957	near 0	1.039	0.325

Table 4.5: Performance of query when rescaling resource provisioning

execution plan	duration diff	CPU diff	physical IO diff	logical IO diff	wait time diff
old plan / new plan	0.403	0.365	division by 0	20,313	0,608

Table 4.6: Comparison of query performance for the different execution plans

As seen in table 4.5 when doubling the amount of resources the performance degraded significantly. This may be a result of the regressed execution plan. If forced to run the old plan it performed better, as expected, shown by the second row.

The difference in performance between the plans at the same resource allocation is quite noticeable. Looking at table 4.6 we can see that the old plan had 60% faster duration while reducing the CPU time by 64%. The new execution plan uses twenty times more physical IO then the old. We also see by looking at first row in table 4.5 a 25 times increase in physical IO which explains why the new plan performs poorly. However, it is also important to note that the runs using the old execution plan uses minimal physical reads. It is possible that the statistics gathered for these two execution plans are outliers due to how efficiently the cache could be used. Where the new plan required most reads to be done from disk and the old plan used the cache almost exclusively this could skew the results of these tests heavily and would need further testing.

While the specifics of the situation may be relevant only to our context, using SQL server. The more important conclusion that can be drawn from this instance is that the underlying statistics that affect the execution plan should be updated in conjunction with a rescaling of resources. Since the amount of available resources,

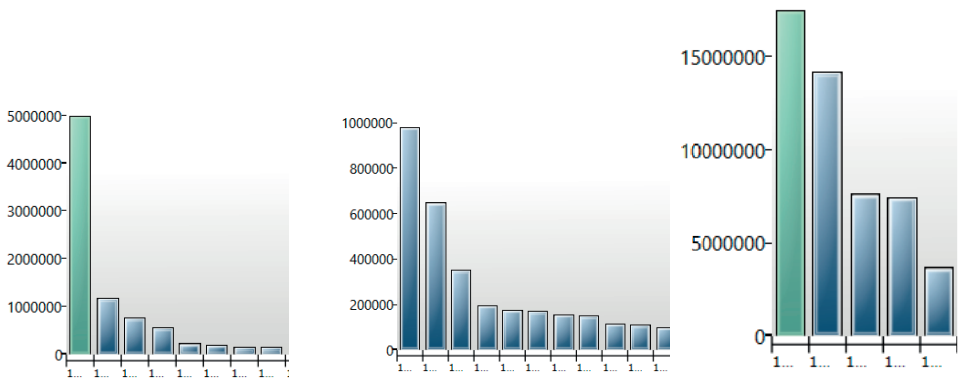
or the change in server impact of a particular execution plan, is a consideration. Execution plan statistics gathered for the previous computing resource allocation may cause problems in the short term. The statistics used will eventually be considered unimportant and a proper execution plan may be chosen granted that the server can keep optimizing this query over a longer time for a given resource provisioning.

4.5 Further Testing of Database Efficiency Metrics

Due to a lack of time, it is not expected that further changes can result in saving costs directly. This would require further analysis of what business critical functions are ran to the different databases and gathering a collection of possible improvements. However, further testing whether database efficiency metrics will consistently provide better performance can still be explored further.

4.5.1 Testing Metrics on the Simulated Production Environment

As the simulated production environment is similar to the development environment, we want to test if changes and observations are also applicable in this environment. First consider which queries have the most impact on the server.



(a) Duration **(b)** CPU time **(c)** Memory consumption

Figure 4.12: The top resource consuming queries, where the highlighted query is most interesting

The top consuming query is the same bulk insert statement that was affecting the development environment. Relatively speaking, it has a larger impact on this environment than what was observed in development. It is run very often and therefore leads to high total resource consumption. As the demanding statement is an insert, it is not interesting to introduce an index to alleviate resource consumption related to this query.

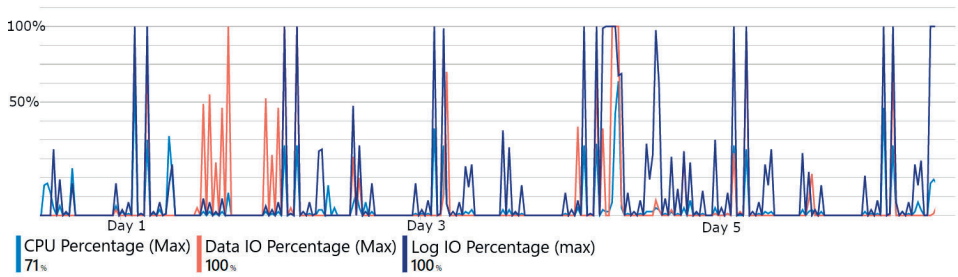
The bulk insert statement is done for a specific table, metrics related to compression are shown in table 4.7

In Row Data	In Row Data	Row Overflow Data	Lob Data
353659	353659	0	0
Compressible Data(%)	**Size MB**	**Read Description**	**Update Description**
100	2762	High Level of Reads	Low Level of Writes

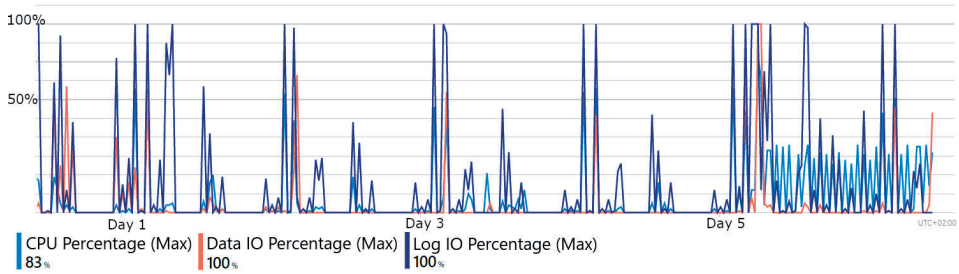
Table 4.7: Table containing metrics regarding compression

Compared to the development environment, it seems that the best course of action is page compressing the primary key for the table. The usage for the different environments vary and can reasonably suggest that different methods are preferred. Following the results from given metrics page compression is applied to the primary key, which is equivalent to compressing the table [20]. Once a compression is done for a table it silently compresses the primary key instead.

Similar to the analysis done for this query on the development environment, canceled and aborted runs are excluded and performance is calculated relative to affected rows. In this case there is an increase of 13% in server processing time, CPU time has decreased by 36% and physical read operations decreased by 46%.



(a) Usage before page compression over 6 days



(b) Usage after page compression over 6 days

Figure 4.13: Utilization difference when compressing in simulated production environment

The utilization graphs, as seen in figure 4.13, are relatively similar. There are less Data IO spikes, and for the weekly batch jobs there is a noticeable increase in CPU usage after compression. The most frequent bottleneck metric is Log IO in this environment. That has not been affected at all by page compressing the table related to the bulk insert statement. By this experiment it is clear that

the varying table usage, data and possibly the current provisioning of computing resources lead to different results for the same database architecture. Restricting methods of optimization to indexing or compression is not sufficient to improve this database in the given environment. Furthermore, the page compression for the same table that had improved performance in a different environment now has worse performance. This means that the database efficiency metrics used to suggest this compression are not applicable without further context of whether Data IO is the constraining factor for performance. Alternatively the performance degradation is a a result of using page compression instead of row compression. If that is the case further metrics regarding compression are needed to make a better decision.

4.5.2 Suggesting Index Using the Database Health Metrics

In order to verify that metrics chosen for suggesting indexes are useful, they need to be tested. The development environment the database for simple application presentation and manipulation contains multiple business critical functions. When attempting to scale down computing resources, and even otherwise, optimizing business critical features ensure that these features will not exceed acceptable response time. This is especially important since this workload can be unpredictable in nature, depending on how the application is used.

By applying metrics regarding indexing, one business critical query was found that would benefit from introducing an index. This query is run every time a user logs into the application and its purpose is to present data to the user. More specifically, the query uses a select statement that contains a sub-query which joins to another table.

group user seeks	query user seeks	equality columns	inequality columns	included columns	column name	column usage
82	16	[Boolean1], [Boolean2]	NULL	[Id]	Id	INCLUDE
82	16	[Boolean1], [Boolean2]	NULL	[Id]	Boolean1	EQUALITY
82	16	[Boolean1], [Boolean2]	NULL	[Id]	Boolean2	EQUALITY
99	16	[Id], [Boolean1], [Boolean2]	NULL	NULL	Id	EQUALITY
99	16	[Id], [Boolean1], [Boolean2]	NULL	NULL	Boolean1	EQUALITY
99	16	[Id], [Boolean1], [Boolean2]	NULL	NULL	Boolean2	EQUALITY

Table 4.8: Metrics about column predicate and seek statistics for query

As seen in table 4.8 by "column usage", the suggested columns for indexing are all used in "where" or "join" statements, which is desired when considering indexing [14]. However all columns except the primary key column are nullable, as

Columns	1	2	3	4	5	6	7	8	9	10	11	12	13
Is Nullable	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 4.9: Columns that are nullable

seen in 4.9. This is not a desirable trait to have when considering indexing. But in this case it might be reasonable to ignore as all the columns that are nullable contain non-null values only. The risk with this reasoning is that the columns

may be filled with null values in the future, but there is no sign in this context of that happening. Lastly the amount of DML operations is investigated. Table 4.10

Proportion of Reads	Proportion Of Writes	Total Read Ops	Total Write Ops
0.98	0.02	21178	329

Table 4.10: Usage of table in question

shows that the operations performed on the table is primarily read operations i.e seeks, scans and look ups. Since the number of DML operations is this small over the table, it is unlikely that the different columns do not conform to similar usage as well. This means that indexes can be introduced according to most suggested metrics for this query and surrounding factors.

After introducing the index, the query duration is 33% shorter, CPU time is 26% faster and wait time decreased by 71%. By examining the execution plan it now uses an index seek instead of a index scan which explains the improvement.

Seek performs better then scan except in two cases, that is if the table is not sufficiently large or if the query uses more then 50% of the table's data [21]. This is not the case for this query as only three of the 13 columns are involved in the query. The size of the table is also large, around 250000 rows.

This is only one result, and it was not necessarily correlated to a reduction in cost, further work would have to be done to verify that the metrics suggested are sufficient to consistently provide significant performance benefits.

Chapter 5
Conclusion

The purpose of this thesis was to discover how to optimize resources on the cloud based on performance and cost savings. All the results gathered were achieved by working systematically as described by our method.

We contribute to this field by developing a systematic approach to observe the performance of a database-as-a-service, in relation to the amount of computing resources, while minimizing its cost. To reach a conclusion we return and answer our research questions.

5.1 Evaluation of Metrics

After usage of the metrics the chosen metrics need to be evaluated based on how useful they were in relation to the original problem statement of what important metrics can state efficient usage of the database. The importance and impact of the chosen metrics were evaluated depending on the category of metrics, or rather their purpose.

5.1.1 Evaluating Performance Metrics

Performance is evaluated relative to changes to the database. Performance metrics should give an idea of how the execution of queries are done and can be correlated to the utilization of the database.

Referring back to the suggested metrics we found most of the chosen ones to be effective, the more evident issue was that the scope first suggested was too slim. Considering metrics belonging to the performance category, server processing time was naturally important as it speaks to the overall efficiency of any given query. However, the details of the processing time became more important as issues related to overutilization showed. Mainly wait time showed large disparities depending on available resources, and should be considered a very important metric when managing cost to performance. On the other hand, availability was rarely a problem that was encountered. While theoretically it would become more frequent when performance is worse, that was rarely the case. This may be a result of the particular context that was experimented with. The timeout window was relatively lenient due to managing the heavy batch queries ran off hours.

Also as we've seen, duration and following issues can be deceiving if certain workloads are ran concurrently. In order to understand the query execution, more independent of concurrency, the duration needs to be split into at least wait time and CPU time. Especially if large concurrent workloads are expected to occur frequently, as is the case with multiple queries belonging to the business critical features in this context. In such a case, it is necessary to consider the relation between performance and utilization at the time of execution.

5.1.2 Evaluating Utilization Metrics

Utilization metrics should be the indicator of whether a database utilization at particular point in time is sufficient. If enough resources are provided performance should not be impaired to the point that execution risks offending QoS.

The utilization metrics, in the form of CPU, Data IO and Log IO were all necessary to give an idea of what the database is bottlenecked by. Optimizing without respect to what metric is currently the problem leads to limited success when optimizing. Using utilization metrics to predict performance however does not seem to be effective. There may be significant changes to how queries are processed as a result of less available resources from execution plans and further waiting that may occur. Storage utilization was rarely encountered in regards to observing the database utilization, since it was never a bottleneck in question.

5.1.3 Evaluating Database Efficiency Metrics

Database efficiency metrics instead need to be evaluated by how consistently they lead to improvements, on performance or utilization, relative to what they suggest. Theoretically, if a situation conforms to the suggested database efficiency metrics for a certain improvement relative to the bottleneck shown by utilization there should always be a performance benefit.

Considering metrics for database efficiency, it was evident that read percent and update percent were important to determine what type of compression to perform. These values can vary depending on the cached workloads however, and ensuring that the statistics for table/index usage is indicative of typical workloads is still important. Determining the amount of compressible data using the types of data storage types is clearly necessary for suggesting compression, in the context of these experiments nearly all data was in row so the evaulation of how much uncompressible data was acceptable was never in question. The suggested metrics did however give one mostly negative result in the simulated production environment. Where the resulting change was lower CPU and Data IO utilization as a tradeoff for longer duration. Results for compression tend to be negative for insert statements as a result of CPU bounds, but this was not found as a problem in the development environment. Even in the simulated production, CPU is not utilized heavily as a result. A possible reason could be the different compression algorithms used for the different environments as a result of given metrics and further experimentation would be needed to separate the changes.

The indexing metrics from the database efficiency metrics were only tested for a single case, but to great effect in the particular case. Corroborating metrics

gathered makes it clear that the predicate over the column is incredibly important. The amount of DML operations performed on the table was rather low. A large amount of DML operations over the included columns would point to issues that may arise outside of the scope of the particular query the index was introduced to solve. Trade-offs can be done depending on context and it can be difficult to define the exact impact that a new index will have for the tables usage as a result. It may also require more maintenance as indexes become fragmented over time. Other column-related metrics such as uniqueness and whether the column is nullable are not quantifiable at this time. For the index introduced, most columns were nullable but never had null values and often these values were relatively unique. The details of what acceptable levels of uniqueness and nullable values remains unknown.

5.2 Discovering Over and Underutilized Resources

Observing persistent time windows of high utilization for resources has been shown to not be enough to predict performance. It is incredibly important to consider the wait times for key functionality. Especially if the functionality is dependant on multiple query results within a short timeframe.

As far as finding an overutilized resource, unrelated to the relative possible resource scalings, the resource should first and foremost be observed in relation to the necessary QoS and how the database efficiency metrics show potential improvements that can be done. If a database does not fulfill QoS and database efficiency seems fine, the database is in need of more resource provisioning. The only limitation in this case is how covering the database efficiency metrics are for further improvements.

Conversely, finding underutilized resources is difficult to strictly define. The initial expectation was that if functions that are business critical, paramount for QoS, require a certain percentage of resources and the database is downscaled to a level that still accommodate these functions without being maximally utilized performance would not be affected. However, results from this paper indicate that when less resources are in effect it is quite common that more reads from disk are necessary, causing more utilization of the CPU to be necessary. Having less available resources indirectly lead to worse performance, whether the database is overutilized or not at the time of execution. Certain optimizations done implicitly by the execution plan may no longer be suitable as a result as well. Wait times can quickly increase, causing significant impact on the performance of certain queries, particularly those that are triggered in parallel with more queries.

Furthermore, a database can be considered underutilized if database efficiency metrics could conclude that a certain performance increase is possible which would lead to downscaling the computing resources without pertinent performance loss. Not only can it be difficult to predict changes in behaviour from the server side with less resources, but the expected performance gain from optimizations are unknown. The optimizations themselves tend to have a certain impact for the targeted query. Following a single, functionally independent optimization, queries that may run in parallel tend to see performance gain since more resources are

available, which affects waiting and execution plans. In the absence of an approximation of optimization impact, the current best course of action seems to be to plan a number of optimizations regarding both the most expensive queries as well as the queries that define the QoS.

5.3 Threats to Validity

The majority of the research was performed on a development environment, which means that it is constantly changing. This in turn can affect the accuracy of certain results if unknown changes are made during data collection. It is also important to acknowledge that the databases tested were used concurrently often. This means that certain data points for certain queries can become inconsistent irrespective of the runtime statistics of the query itself, but rather the workload placed on the database at the particular time frame. To eliminate these issues, one would need to test the metrics in a more contained environment.

The databases explored were quite large and the workloads unpredictable, which complicates introducing indexes and analyse the many different effects they may have. In this case, indexes were examined as they related to a specific query, where one of the database efficiency metrics related to whether the newly implemented index would be involved in multiple DML operations. Within the large system this context provides, it will be difficult to pinpoint all of the effects that a index has, both over time and for usage in unexpected situations.

The cloud platform constrains the ability to scale resources freely. Specific change intervals in resource allocation makes it harder to collect more information regarding different behaviour at more detailed provisioning. This limits the possibility of finding a scaling option that is most suitable for the workload.

Certain time intervals of collecting data are rather short, and especially for certain behaviour that occurs at the end of the week or month. Most time intervals for changes span a week or two. This makes filtering anomalies in statistics difficult to separate as well as determining consistent improvement over a longer time.

When testing changes for different resource allocations for the databases, these experiments have mostly compared relative changes to server processing time and wait time. Eventual relationships between the different aspects of the server processing time has not been explored, which may give a deeper understanding of the execution. Cloud related issues that may affect execution as a result of a larger unknown infrastructure and communication was not available at the time of experiments. For this reason, server processing time was the most studied duration of the response time, where the other may also have a large impact of the performance demands set by the QoS.

5.4 Future Work

We would like to further test these metrics in multiple contexts with varying workloads. More variations of workload would then test the correctness of said metrics and the following optimizations. As the platform was quite restrictive during these experiments. Further options in scaling of resources would support

more investigations of breakpoints in over and under utilization. This would lead to clearer and more concise conclusions regarding the states defining the difference between over-/under utilized resources.

As these metrics, and others that may be missing from this report, become better defined the process of observation and optimization becomes better defined as well. Some metrics for a more thorough examination of execution are currently not possible to monitor in the given environment. Further exploring the different aspects time spent in the CPU and waiting could provide more crucial patterns of under and overutilized databases.

The potential for adding more database efficiency metrics for other improvements would benefit the understanding of how the database may be currently inefficient and what potential solutions are available. Using that, costs could more effectively be saved for similar situations.

As a result of the compound offer of CPU, Data IO and Log IO compression became an invaluable tool to alleviate problems for Data IO utilization in exchange for higher CPU utilization. If compounding resources in this way will continue being the premier way of acquiring processing power specializing in solutions that can alleviate Log IO and CPU utilization may lead to particularly important cost reductions for this kind of environment and should be explored.

Another interesting question is if one could gauge how much gain in performance can be achieved by performing an optimization. In our case the potential gain from optimizations where unknown as metrics only suggested if a optimization would bring an expected improvement, but they cannot quantify how much. Understanding how much the performance could be better, and particularly the impact it has on concurrent usage, is a crucial aspect of saving costs as a result of optimizing aspects of the database.

References

[1] O. International, "The three service models of cloud computing." https://www.openintl.com/the-three-service-models-of-cloud-computing/#:~:text=Cloud%20computing%20is%20offered%20in,as%20a%20Service%20(IaaS).

[2] L. Shui F. and C. K. Hung, *Computer Capacity Planning.* Academic Press, n.d.

[3] J. Shao and Q. Wang, "A performance guarantee approach for cloud applications based on monitoring," in *2011 IEEE 35th Annual Computer Software and Applications Conference Workshops*, pp. 25–30, 2011.

[4] W. Wu, Y. Chi, H. Hacígümüş, and J. F. Naughton, "Towards predicting query execution time for concurrent and dynamic database workloads.," *Proceedings of the VLDB Endowment*, vol. 6, no. 10, pp. 925 – 936, 2013.

[5] T. Westmann, D. Kossmann, S. Helmer, and G. Moerkotte, "The implementation and performance of compressed databases.," *SIGMOD Record*, vol. 29, no. 3, pp. 55 – 67, 2000.

[6] Microsoft, "Row compression implementation," June 2021. https://docs.microsoft.com/en-us/sql/relational-databases/data-compression/row-compression-implementation?view=sql-server-ver15.

[7] Microsoft, "Page compression implementation," December 2020. https://docs.microsoft.com/en-us/sql/relational-databases/data-compression/page-compression-implementation?view=sql-server-ver15.

[8] A. Abbas and K. Ahmad, "An integrated indexing approach to improve query processing.," *2021 International Conference on Electrical Engineering and Informatics (ICEEI), Electrical Engineering and Informatics (ICEEI), 2021 International Conference on*, pp. 1 – 6, 2021.

[9] Microsoft, "Clustered and nonclustered indexes described." https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-ver15.

[10] Microsoft, "Clustered index structures." https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms177443(v=sql.105)?redirectedfrom=MSDN.

[11] Microsoft, "Nonclustered index structures." https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/ms177484(v=sql.105)?redirectedfrom=MSDN.

[12] P. Bozzelli, Q. Gu, and P. Lago, "A systematic literature review on green software metrics," *VU University, Amsterdam*, 2013.

[13] M. Valavala and W. Alhamdani, "Automatic database index tuning using machine learning," in *2021 6th International Conference on Inventive Computation Technologies (ICICT)*, pp. 523–530, 2021.

[14] M. Valavala and W. Alhamdani, "A qualitative case study of relational database index tuning using machine learning," in *2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, pp. 389–396, 2021.

[15] B. Ball, "Introducing what_to_compress v2," May 2014. https://www.sqlservercentral.com/blogs/introducing-what_to_compress-v2.

[16] S. Mishra, "Data compression: Strategy, capacity planning and best practices," May 2009. https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008/dd894051(v=sql.100)?redirectedfrom=MSDN.

[17] S. Das, F. Li, V. R. Narasayya, and A. C. König, "Automated demand-driven resource scaling in relational database-as-a-service," in *Proceedings of the 2016 International Conference on Management of Data*, pp. 1923–1934, 2016.

[18] S. Ewen, M. Ortega-Binderberger, and V. Markl, "A learning optimizer for a federated database management system.," *Informatik Forschung und Entwicklung*, vol. 20, no. 3, pp. 138 – 151, 2005.

[19] Microsoft, "Query processing architecture guide," December 2021. https://docs.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-ver15.

[20] Microsoft, "Creating compressed tables and indexes," April 2012. https://docs.microsoft.com/en-us/previous-versions/sql/sql-server-2008-r2/cc280449(v=sql.105)?redirectedfrom=MSDN.

[21] C. Qi, "On index-based query in sql server database.," *2016 35th Chinese Control Conference (CCC), Control Conference (CCC), 2016 35th Chinese*, pp. 9519 – 9523, 2016.

Printed by Tryckeriet i E-huset, Lund 2022

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2022-886
http://www.eit.lth.se