PREDICTING SQL EXECUTION TIME

Ravikiran Durbha

September 20th, 2016

Background and Description of Problem

In any large transactional system there will be many access paths to retrieve a given dataset. The RDBMS optimizer makes a heuristic-based decision on the best access path for a query at compile time. There are many factors that could influence this choice, such as the system resources available (I/O, CPU and memory) and the objects available in the database such as indexes (clustered and non-clustered), views and materialized query tables etc. The access path determines the execution time of the query. Another important factor that determines the execution time is the workload present in the system at the time of execution, as this would determine availability of some system resources.

One of the challenges in developing on such a transactional system is to ensure that queries will have the best possible throughput which is determined by all the factors described above. The faster the queries execute the higher is the throughput of the whole system. One way to solve the problem is by providing a clone of the production system to the development system with exactly the same system resources and database objects. There are a couple of drawbacks in this approach:

- this could become expensive depending on the class of hardware and volume of data.
- it is extremely difficult to mimic the production workload on the cloned system.

We may be able to overcome the disadvantages of the above solution by training a statistical model to predict the performance of the queries given execution data about prior queries on the production system

The approach taken to measure the workload was to take the average of the workload metrics over a period of one hour just prior to the period in which we are looking to predict the execution time of a given query. In other words, if we want to predict the execution time in the period 12PM – 1PM, we input the average of the workload metrics from 11AM to 12PM. The workload in the system in question is pretty stable across the working hours. This approach may not be effective if the workload in the system is very volatile.

Data and description of features

The RDBMS engine provides many metrics to monitor the activity in the system. We are interested in the metrics that impact the execution time of the query. These metrics can be collected at many different levels, such as the application level, the individual SQL level or the database level. Since the purpose is to model the average workload of a database system one hour prior to the execution of a query, the data was collected at the database level.

Let us look at some of these metrics:

Lock wait time:

Each time an application needs to access a table it needs to acquire a lock, so that no other query updates any record in the table while the first query is using it. If a lock is requested on a table when another application is already holding a lock, then the lock requesting application is kept in a wait state until a lock can be acquired or a time out threshold is reached (configured to be 10 s in the system where the data is being collected from). This will impact the execution time of the query if the application executing the query has to wait to acquire a lock. This metric is a measure of the amount of overall time the RDBMS engine spent waiting for locks. This is accumulated from the time the engine started.

Total section sort time:

This metric represents the time the RDBMS engine spent in fulfilling sort requests. Sorts not only take up CPU cycles as it parses pages in memory, but it could also impact I/O as data spills onto disk. The assumption is that the amount of sorting being done on the system could indicate prospective demand for the resources and hence impact the execution time.

Diagnostic log write wait time:

Depending on the level of diagnostics requested from the RDBMS engine, it could delay the execution of a query if there are issues during the execution that need to be logged. This metric is expected to have least correlation with the execution time as in most cases if there are issues that warrant logging, the SQL execution would result in a failure.

Direct read time:

This metric is an aggregation of all the time spent by the RDBMS engine in reading the data directly from disk without going through the operating system. This happens when the RDBMS has to read a large volume of data (for example during backup operation or large binary objects). This usually indicates a high I/O load and could impact the execution time of the incoming SQL query.

Direct write time:
This is similar to the metric above except that it is for writing.

Log buffer wait time:
The RDBMS engine performs write ahead logging -- updates/inserts are first written to the log buffer before they are actually written to the data page in memory -- and if the log buffer is full the write has to wait for the buffer to be flushed. Each time an application has to wait for the log buffer flush, the log buffer wait time counter is increased. This metric only impacts write operations so we expect this metric to have minimal impact on the read operations.

Log disk wait time:
This is similar to the metric above, except that it measures the wait incurred due to I/O wait.

Pool read time:
This metrics represents the total amount of time the RDBMS spent in reading data from the buffer pool. This impacts all queries as this operation is performed for all queries.

Pool write time:
This is the write counterpart of the above metric and should only impact the write queries.

Prefetch wait time:
When the RDBMS has to perform sequential read/write (basically read/write a lot of records) then, to improve performance, it asynchronously prefetches data and this metric captures any time spent by the RDBMS waiting for such an operation to complete. This should mostly impact bulk read/write operations.

Total Activity time:
This is a measure of the time the RDBMS engine spent in all the activity (including the ones listed above). It also includes some activity that is not included above.

Total Activity wait time:
This is a measure of all the time that RDBMS spent in waiting for different purposes, including the some of the metrics listed above.

Total CPU time:
This is a measure of the total CPU cycles used by the RDBMS engine.

Total extended latch wait time:
This is the total time the RDBMS engine spent in waiting for latches, a type of internal lock that the RDBMS engine uses to serialize resource access.

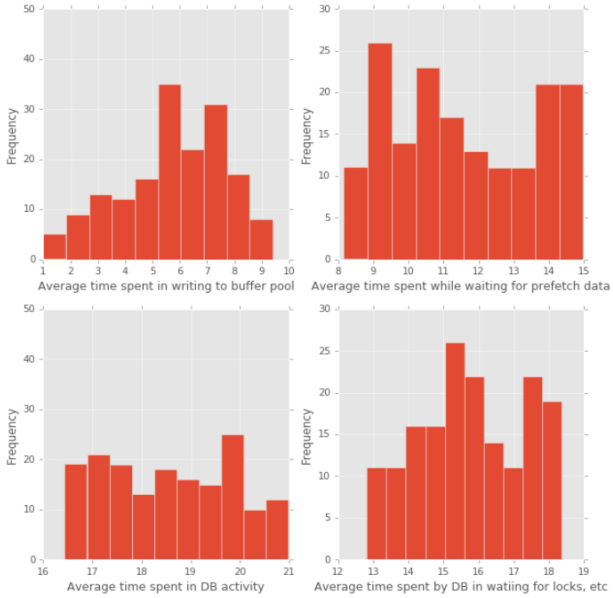
The RDBMS engine aggregates the metrics continuously and we snapshot the data every five minutes and compute the difference between the current and the previous snapshot to derive the value of the metrics in that duration and then compute the average in the one hour period prior to the SQL execution window to derive the workload metrics.

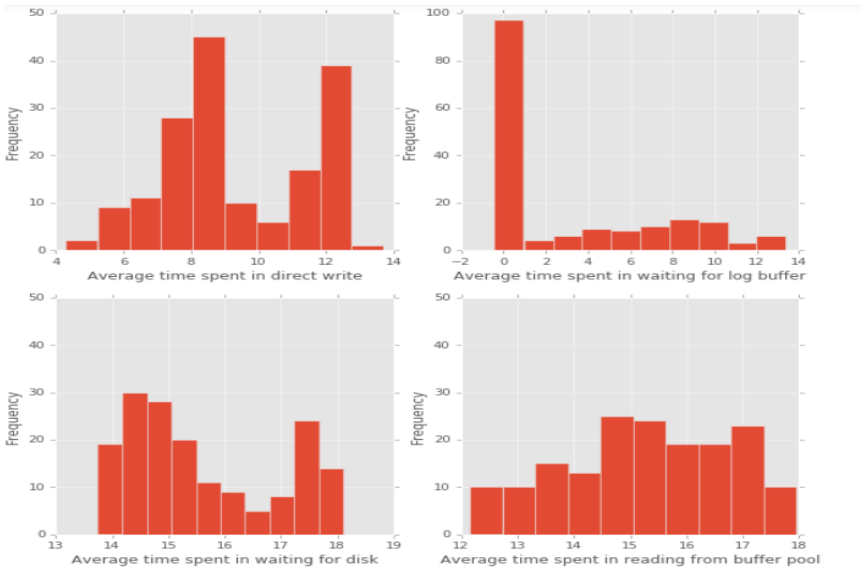
The total time metrics were included due to the fact that in addition to the components listed above, they also include other components for which the individual times could not be captured or not provided. It needs to be investigated to determine if the total time metrics have any correlation with the individual time components.

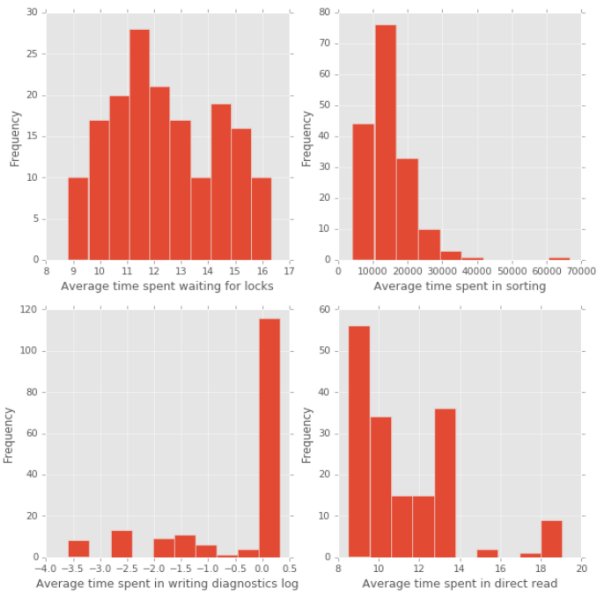
Feature selection and analysis

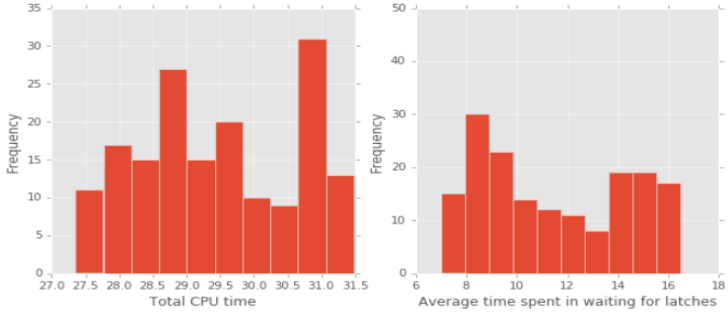
The features were log-transformed to reduce the scale of the distribution, hence the regression coefficients are to be interpreted as a change in the response variable for every doubling of the independent variable [1].

The distribution of many of the features displayed bimodal characteristics, suggesting different classes of queries. Some of the features did not display a normal distribution such as diagnostic log write wait time and log buffer wait time. These variables were retained to analyze the correlation and regression coefficients.

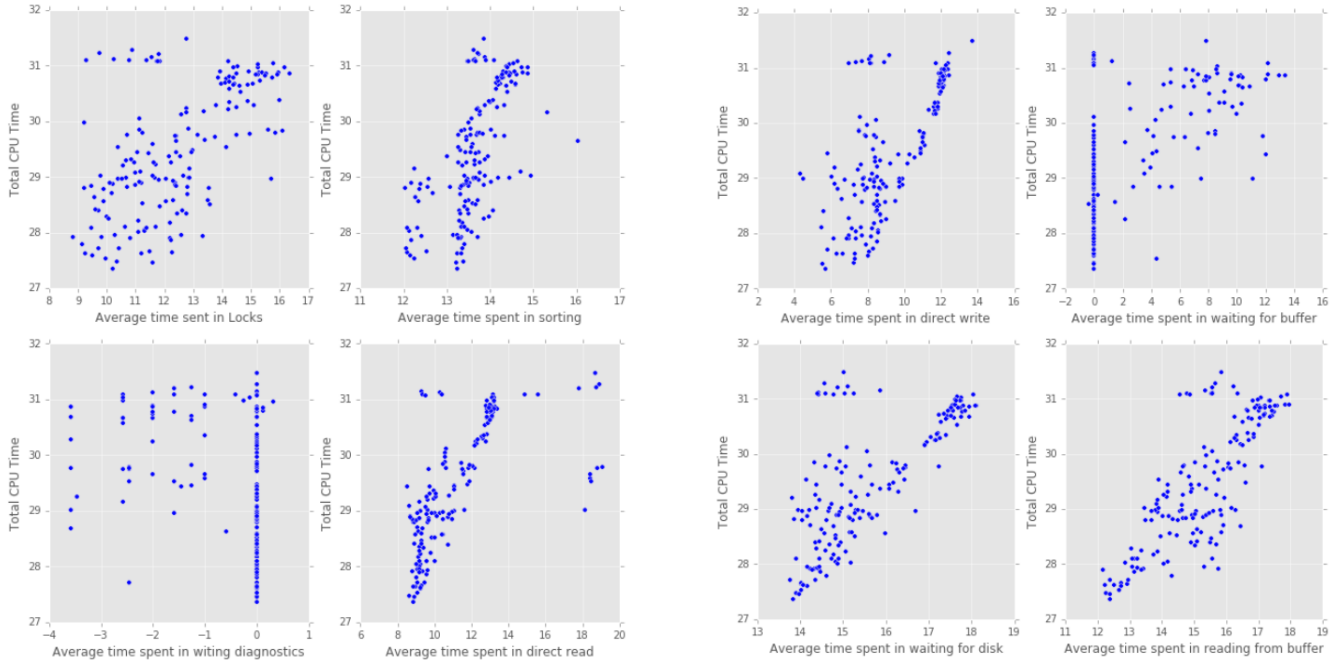


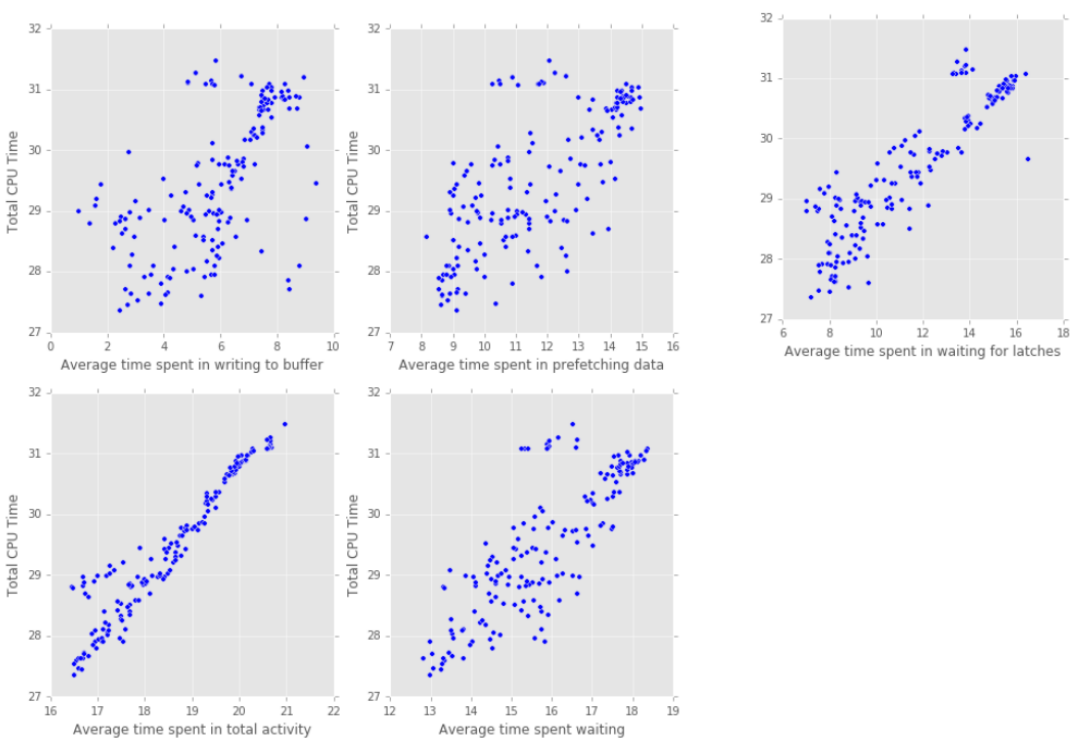




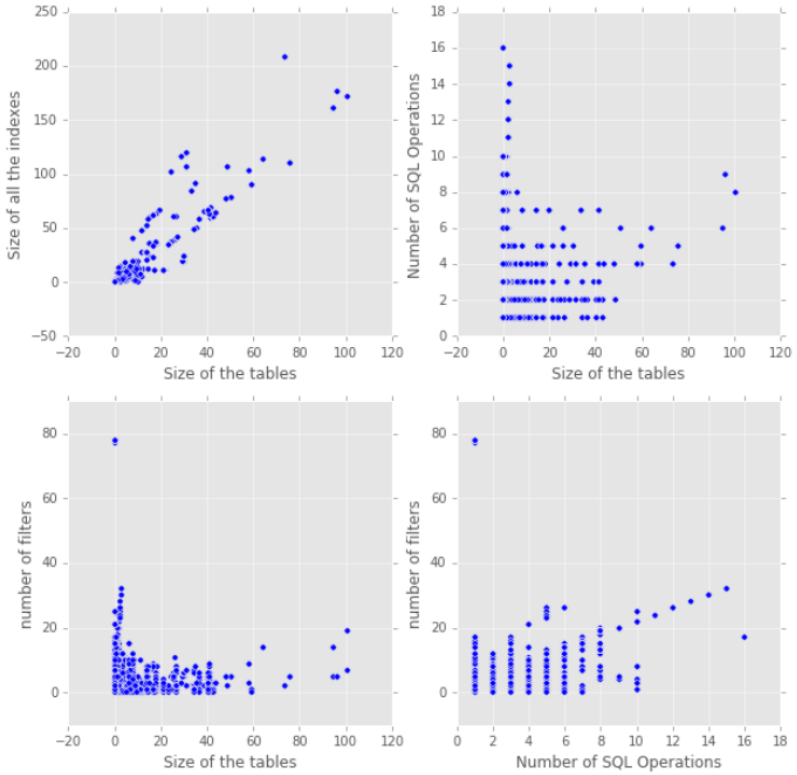


We looked at the correlation between the aggregate workload metrics such as CPU time and other activities to see if there is any strong correlation that we can identify. While there was some correlation between CPU time and some metrics, we did not notice anything so strongly correlated that warrants not considering that metric.





Aside from the analysis on the workload metrics, features extracted from the actual queries were also analyzed. Strong correlation was noticed between operand size and index size, which was expected as larger tables usually have larger indexes. As a result, it was decided that index size was not adding any new information to the model and that it could be discarded. The scatter plots are included below.



Regression analysis

There were many considerations to ensure that we develop a model that will predict the response variable (execution time) with a fair amount of precision. We investigated the following parameters to assess fit.

Coefficient of determination (R-Squared):

The R-squared value indicates how much of the variance in the response variable is explained by the model. It is expressed as a percentage. While having a high R-squared value indicates that the model explains the variance in the response variable, it may not generalize very well to future data due to overfitting. Hence we consider R-squared along with other parameters to ensure the model is predictive.

Additionally, the R-squared tends to increase as we include more features, but a higher R-squared value does not tell us if it is a random change in the variation or the newly added parameter actually explains the variation in the response variable better. We can look at the adjusted R-Squared which will increase only if the model explains the variation by more than just chance. We can use this to determine if the addition of a feature improves the model fit. So we start with one feature and start adding other features one-by-one and see if the adjusted R-squared increases or not. If it does not, then that feature did not improve the model . We can see this method in action below:

We start with total CPU time, since it is an aggregate of all the activity in the system:

Features Included : Total CPU Time

Parameter	Value
R-Squared	0.874
Adj. R-Squared	0.873

As we see the model above with one feature actually explains 87.3% of the variation in execution time. Now, we start adding other features and notice how adjusted R-Squared changes:

Features Included: Total CPU Time , Lock time out

Parameter	Value
R-Squared	0.875
Adj. R-Squared	0.874

We can see that the adjusted R-Squared value did increase indicating that the change in the variation explained by the new variable is not by chance. This does not mean we automatically include this feature. It remains to be seen if the coefficient is statistically significant by analyzing the t-value and its significance for each of the features. Now, we add more features to see how the Adjusted R-squared value changes.

Features Included: Total CPU Time , Lock time out, Sort time

Parameter	Value
R-Squared	0.875
Adj. R-Squared	0.873

As can be seen in the table, the adjusted R-squared value actually decreased, indicating that adding the new feature did not yield a model with a better fit. So we remove it and add other features. We got pretty much the same result adding other workload metrics. It was pretty clear that we were not getting a better fit by adding any more additional workload features. So now we add SQL metrics to see what impact it has on the Adjusted R-Squared value. Features were also added in different order, but the results were similar.

Features Included: Total CPU Time , Lock time out, Operand Size

Parameter	Value
R-Squared	0.887
Adj. R-Squared	0.884

We can see that the adjusted R-Squared value increased and we are able to explain 88.4% of the variation in execution time with these features. We continue to add other SQL metrics. When we finally added all the SQL metrics and the model was able to explain 88.8% of the variation, which is the highest we achieved.

Features Included: Total CPU Time , Lock time out, operand size , number of filters , number of operations

Parameter	Value
R-Squared	0.892
Adj. R-Squared	0.888

Significance of Coefficients (t-statistic and p-value):

The p-value of the t-statistic tests the null hypothesis – Is the coefficient of the feature/variable zero? If the p-value is below the threshold (usually 0.05 or 5%), then we can reject the null hypothesis and assume that the computed coefficient is statistically significant. The t-score is computed for every feature keeping the other features constant. Hence, this indicates the significance of each of the coefficients individually. The overall significance of the model is measured using the f-test and its corresponding p-value.

We look at the t-score for each of the features below :

Feature	Coefficient	t-score	p-value
Total CPU time	0.0822	2.970	0.003
Lock Time Out	1.54e-06	0.688	0.492
Operand Size	-0.3358	-4.255	0.000
Number of Filters	1.0168	2.317	0.022
Number of Operations	-1.4015	-2.486	0.014

As we can see, the p-value for all the coefficients are below the threshold of 0.05, except for “Lock Time Out”. We can also see that the coefficient for the “Lock Time Out is very close to 0 and the t-score further indicates that it is not statistically significant. We can discard this feature from the model.

Once we remove the “Lock Time Out” variable, the results for the t-score and p-value are as below:

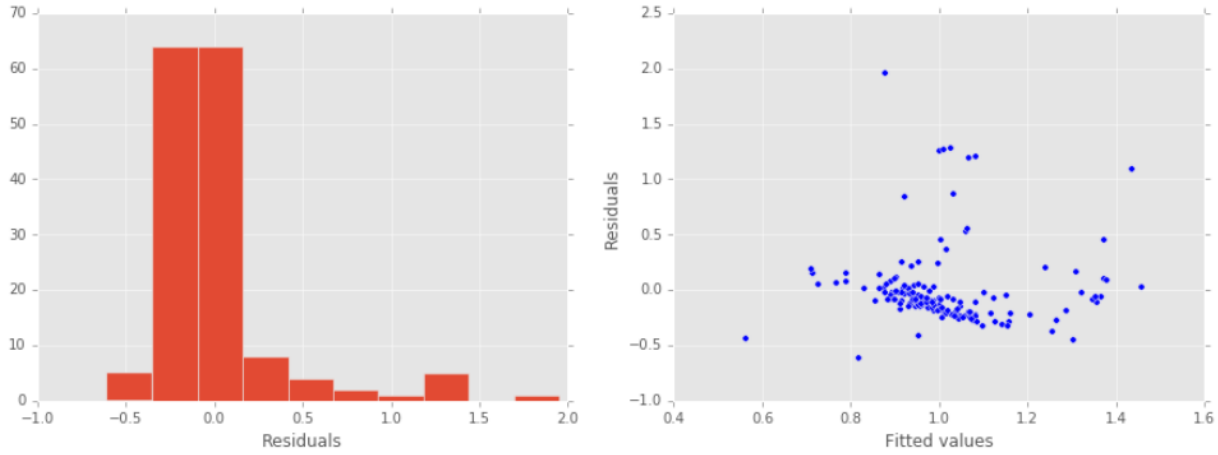
					95% Confidence Interval	
Feature	Coefficient	Standard Error	t-score	p-value	Low	High
Total CPU Time	0.0872	0.027	3.271	0.001	0.035	0.153
Operand size	-0.3076	0.067	-4.566	0.000	0.140	1.884
Number of Filters	1.0181	0.438	2.324	0.021	-0.441	-2.578
Number of Operations	-1.5108	0.540	-2.797	0.006	-0.175	-0.444

As we can see, all of our features are now statistically significant. Additionally, we also achieved a higher Adjusted R-squared value without the “Lock Time Out” feature.

Parameter	Value
Adjusted R-Squared with “Lock time Out”	0.888
Adjusted R-Squared without “Lock time Out”	0.889

Residual Plots:

Residuals, or the errors in the regression, are expected to be random and not display any pattern. If they do, it could indicate that the model is failing to account for all the deterministic variance in the data. In general, the errors are supposed to capture the noise which should be random and stochastic. Also, the errors should be normally distributed. Looking at our residual plots no specific pattern can be identified and we can see that the distribution is more or less normal.

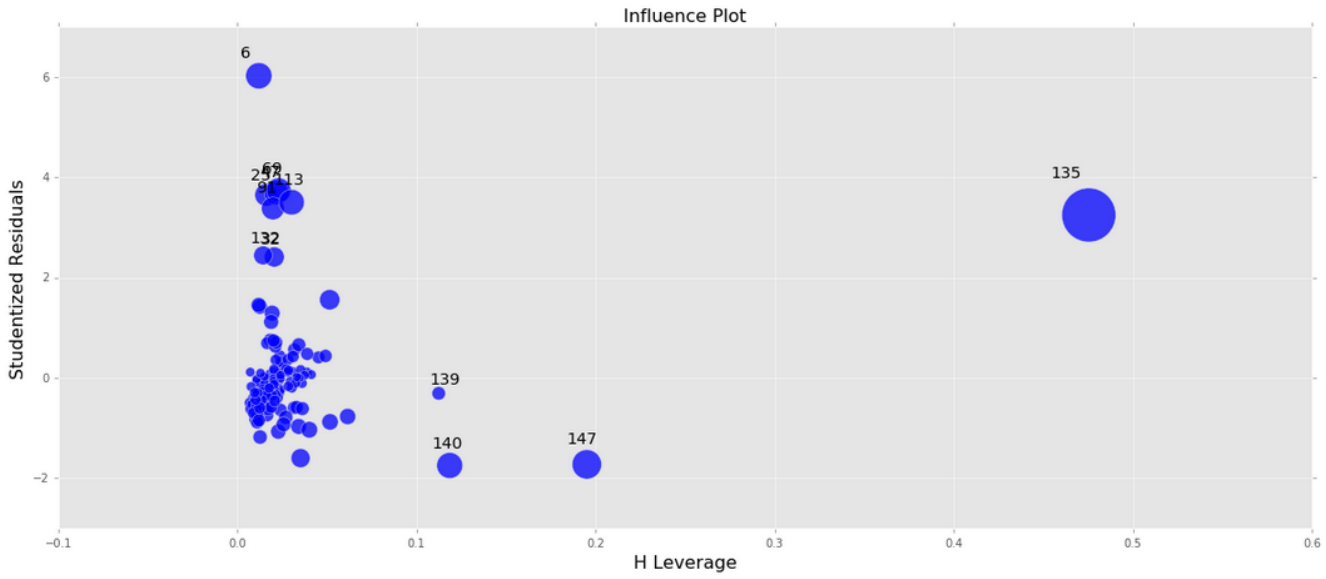


Influence Plot:

The influence plot gives us a visualization of how much each observation influences the fit of the regression line. Two factors impact the influence:

1. How much the observation's value on the predictor variable differs from the mean of the predictor variable. This is the leverage of the observation
2. The difference between the predicted score for the observation and its actual score. This is the distance of the observation.

The influence graph below plots the leverage on the x-axis and the distance as studentized residuals on the y-axis. The points that have both high leverage and large distance exert high influence on the regression model.

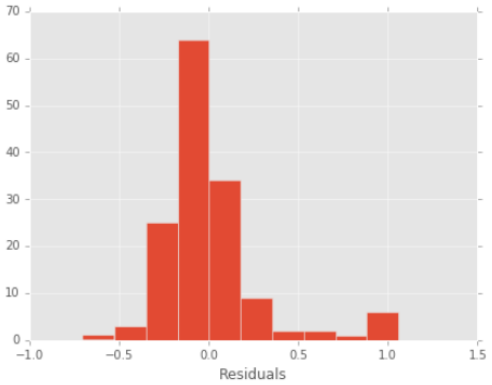


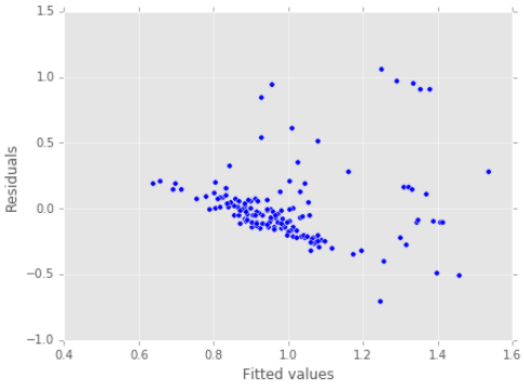
As the plot shows, there are some observations that are influencing the regression model much more than other observations. These high influence points can be removed to see if the model improves overall and if it generalizes the data better to improve prediction. Regression was performed again without the high influence points (6,135,147,140) as can be seen in the plot. The result of the regression is summarized below.

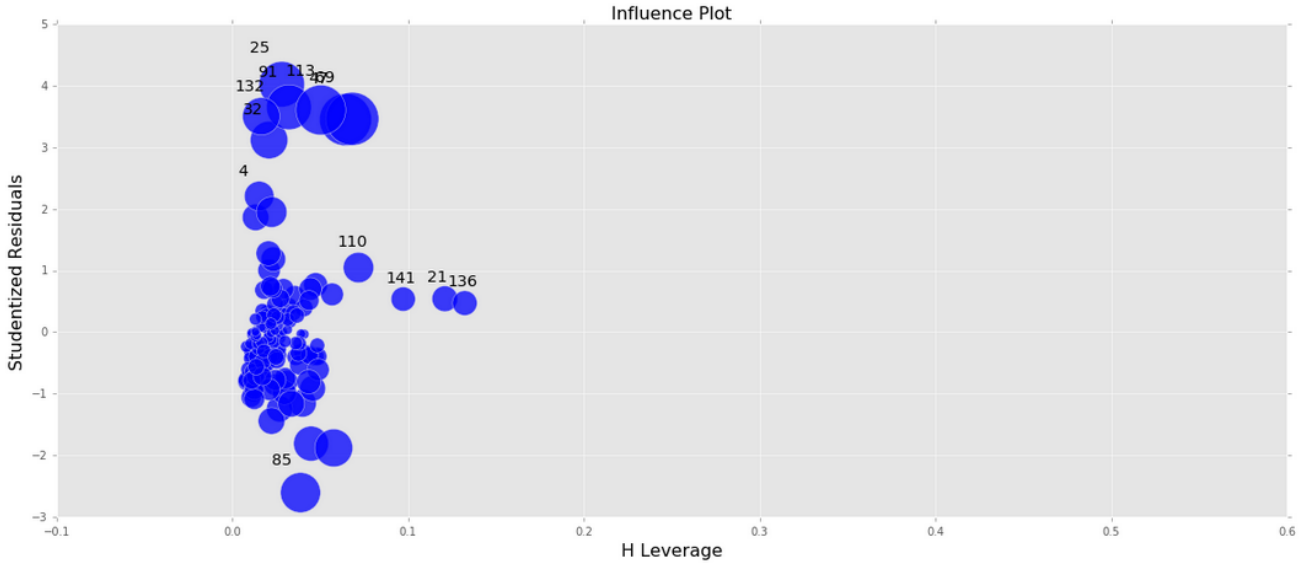
It should be noted that the adjusted R-squared went up to 93% which indicates that we are able to better explain the variance after omitting the outliers and high leverage points. We were only able to explain 89% of variance with the outliers included. We can also see that the residual distribution is much closer to normal distribution and no specific pattern emerges in the residual plot. Additionally, the influence of the observations is better distributed and not concentrated on just a few of them.

					95% Confidence Interval	
Feature	coefficient	Standard Error	t-score	p-value	Low	High
Total CPU Time	0.1127	0.022	5.151	0.000	0.069	0.156
Operand Size	0.2458	0.104	2.354	0.020	0.039	0.452
Number of Filters	-4.0137	0.867	-4.629	0.000	-5.728	-2.300
Number of operations	2.5848	0.816	3.166	0.002	0.971	4.199

Residual Distribution and influence plot(without the outliers):







Conclusion

The analysis above shows that there is a relationship between the SQL metrics like number of operations, size of the tables, and number of filters with the execution time. This was certainly expected as all those metrics would impact the amount of time RDBMS would take to execute a query. What was not very clear at the beginning of the analysis was the relationship between the workload metrics and the execution time of the query. A total of twenty workload metrics were initially identified as candidate features in the model that could help predict the execution time.

Further analysis made it clear that the affect of workload on the execution time can be captured in the average CPU time consumed by the RDBMS in the prior hour. The other workload metrics were not improving the fit of the model.

To be specific, the relationship between the CPU time and the SQL metrics with the execution time was estimated to be :

Execution time = 0.1127 * (average CPU time) + 0.2458 * (Table size) – 4.0137 * (Number of Filters)
+ 2.5848 * (Number of operations)

As can be seen in the model above that the Number of Filters is inversely proportional to the execution time, which is expected as the total number of filters increase the amount of data on which the operations have to be performed decreases, hence decreasing the execution time of the query.

The model was used to predict execution time on unseen data and the table below shows the sample of the result of the test.

ID	Query	Execution Time (ms)	Predicted Execution Time (ms)	Error
1	UPDATE Fulfillment.ServerIndexStatus_2 SET D...	0.04	0.34	-0.30
2	SELECT GroupId, ServerName, DatabaseName, Gene...	0.13	0.34	-0.21
3	SELECT ResponseId,EventId,SupplierId,PrebidRes...	0.03	0.05	-0.02
4	SELECT FolderId, UserId, DelegatedByUserId FRO...	0.06	0.05	0.01

Mean squared error of the model was computed for both the training data and the test data to measure the performance of the model on unseen test data. The mean squared error on the training data was 0.02 and the mean squared error on the test data was computed to be 0.09. There were approximately 120K queries in both training and testing data.

As a future scope, the analysis could include predicting system resources the SQL execution will consume. This will enable optimization in the cloud as infrastructure is purchased as utility. Additionally, prediction could be extended to a set of SQL statements or a given workload instead of individual statements. Since in many cases SQL statements are executed as a workload or a set and not independently. This will make it more adaptable to different situations.