Prediction Models for Resource Utilization in Teradata

Vishal Joshi
University of Michigan
vmjoshi@umich.edu

Kritika Versha
University of Michigan
kmversha@umich.edu

ABSTRACT

Resource utilization estimation is becoming increasingly important, particularly in the context of OLAP workloads. It is a diffult problem because performance in traditional DBMS engines depends on many factors such as physical storage layout, predicate selectivity, relation cardinality, indexing structures, and join combinations and algorithms. Performance fluctuations occur due to complex interactions within the engine where multiple components compete for resources and consistency guarantees must be provided. Teradata provides robust logging features to help database administrators analyze these parameters, but there are no satisfying solutions for automating this information processing and dynamically adapting to new workloads.

While performance analysis may appear daunting, it is critical for administrators to identify which queries are hogging resources and whether the workload can scale with a growing dataset. In this paper, we present an interactive tool for predicting resource utilization of incoming queries by developing regression models with a small training set on the target workload and a fixed schema and dataset. It provides estimates for latency, CPU usage, and I/Os to assist administrator planning. We evaluate different models on the TPC-H benchmark and provide recommendations for tuning input parameters.

Categories and Subject Descriptors

H.4 [**Systems**]: Relational databases

Keywords

OLAP, Performance Predictions, Machine Learning, Resource Utilization

1. INTRODUCTION

With commercial database sizes approaching the scale of petabytes and the growing trend of âĂIJbig data,âĂİ it is becoming increasingly difficult to tune timing and resource usage parameters for OLAP workloads that are characterized by ad-hoc, read-mostly queries that end up performing several sequential disk I/Os scanning entire relations. Database administrators end up spending a lot of time investigating which queries are hogging resources and decreasing overall throughput. Without properly constructed models and techniques to perform this analysis, attempting to predict these critical parameters can be a shot in the dark, especially in complex distributed systems that compute in parallel. A better solution is required for examining workload behavior under a fixed dataset and schema.

Prediction is at the heart of almost every discipline, and the study of generalized prediction from data is the central topic of machine learning and statistics, and more generally, data mining. Knowledge discovery in databases is attracting a lot of attention due to a an increasing amount of data being collected and analytical activity on rise. In this paper, we present different regression models from statistical machine learning applied to databases to predict the performance and resource utilization which would help a lot of businesses using vast amount of data to perform analysis on their dataset. Our focus mainly lies on parallel databases which can be used for various kinds of knowledge discovery activities.

2. TERADATA ARCHITECTURE

Teradata has a foundation based on the *Shared Nothing* architecture. The main components of the Teradata architecture are its Parsing Engine, Access Module Processor (AMP) and BYNET. The parsing engine is also called the Optimizer. Teradata makes use of the Optimizer to make the best decisions possible about resource utilization using the database statistics. It uses statistical data that has been generated using different modules and log tables, utilizing the Query Capturing Facility to optimize the query. The Optimizer then passes these query plan statistics to the AMP along the BYNET communication channel. The AMP is the core of this parallel database and has its own disk to read/write capabilities, thus enabling parallelism. Figure 1 illustrates the architecture, demonstrating the interaction of the parsing engine with multiple AMPs.

3. FEATURE EXTRACTION

To facilitate workload analysis, Teradata provides rich logging features for collecting statistics on queries and query plans. We examined three of these log tables to identify features that would yield the best predictive accuracy when in-

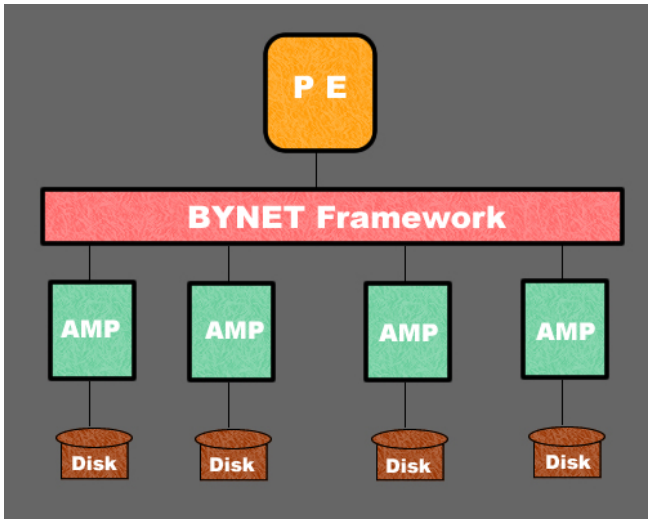

Figure 1: The parsing engine (PE) uses BYNET to communicate with the AMP nodes.

put to the regression models. Each table provides different statistical information, requiring us to study the tradeoffs between highly desirable features and usability in terms of on-the-fly extraction.

3.1 Query Capture Facility

The Query Capture Facility (QCF) stores the steps in a given query plan in a set of predefined relational tables called the Query Capture Database (QCD). The principal source of the captured information in QCD is the white tree produced by the Optimizer, the same data structure used to produce EXPLAIN reports. While the QCD XML table that stored details about the query plan was attractive for feature extraction, especially since it is the only logging feature that does not require the query to actually be run, we were not able to capture sufficient features from it to provide a satisfying model.

3.2 DBQ Log Table

This table is the key source of data to support application performance and workload management analysis. It helps identify the performance resource utiliztion of queries. The table logs the three output parameters which we used our models to predict: query response time, total I/O count, and AMP CPU time.

3.3 XMLPLAN

The XMLPLAN table combines both the DBQ Log Table statistics with a larger set of parameters for the query plan than provided in the QCD XML table. However, it does not log EXPLAIN commands so the query needs to be run for the XMLPLAN table to be populated. We extracted features and output parameters by joining the XMLPLAN table and the DBQ Log Table on the unique query IDs associated with our training and test sets.

4. MODEL PARAMETERS

Our key contribution is determining a set of features to provide accurate predictions of important statistics such as

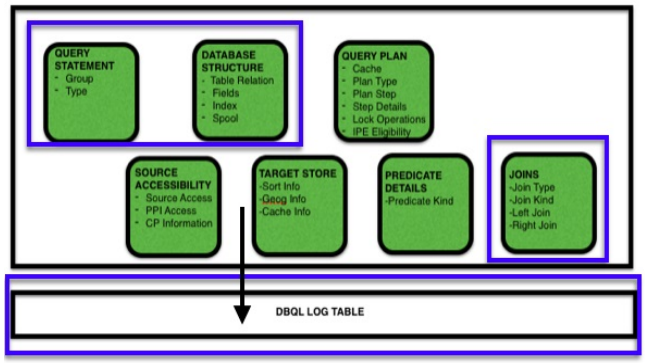

Figure 2: The DBQ Log Table extracts statistics about the query text, the query plan, and the database structure.

query response time, CPU usage, and disk I/Os for a query. We analyzed the Teradata architecture and identified features that are strongly tied to the performance of a query. The following components of the Teradata DBMS gave us insight on which features to select.

4.1 Access Module Processor

Each AMP is an instance of the database management software responsible for accessing and manipulating data. Usually, the AMP obtains its portion of the disk space by being associated with a virtual disk (vdisk). It handles its disk reading/writing by using its file system software, which converts AMP steps (i.e., the steps from the parsing engine) into physical data block requests. The AMPs are responsible for accessing and manipulating the data so as to complete the request processing. There may be mutiple AMPs on one node, and the communication among the AMPs is controlled by the BYNET framework.

4.2 Query Structure

Depending on the number of tables or fragments that a query must search and the resources that are available for a decision-support query, the database server assigns different components of a query to different threads. Different levels of locking, types of joins, sorting, and aggregations impact the response time and hence the performance of the database. Extracting features from the query text itself is a difficult problem and outside the scope of this paper.

4.3 Indexing

Creating the right indexes ensures that the maintenance will be effective, and an appropriate design will simplify plans and even make specific indexing features available. Performance problems can be worked on using indexes. The EXPLAIN modifier is useful in determining whether the indexes defined for a table are properly defined, useful, and efficient. IndexMaintenance in the Query Capture Facility contains one row for each SQL statement / index combination where maintenance costs are required for the index. The different types of indexes used along with its estimated cost and space provides useful information to capture the performance of resource utilization.

4.4 Joins

Since hash joins and large sorts are memory-intensive operations, using these features would let the regression model understand and track the amount of memory it can use. Depending on the number of tables or fragments that a query must search and the resources that are available for a decision-support query, the database server assigns different components of a query to different threads.

5. OUTPUT PARAMETERS

We chose three output parameters that give a holistic overview of the resource utilization of a given query. These parameters were extracted from the DBQ Log Table.

5.1 Query Response Time

One of the problems administrators face is determining exactly *where* the database or user sessions have been spending their time. Given all the possible activities and interactions within the database, these tasks are far from trivial. The query response time helps in deciding if your database is experiencing a high percentage of waits and bottlenecks due to a particular query.We determining the response time by finding the difference between the starting time of the query and the time of the first response observed by the query.

5.2 Total I/O Count

A data block is a disk-resident structure that contains one or more rows from the same table and is the smallest I/O unit for the Teradata Database file system. Data blocks are stored in physical disk sectors or segments, which are grouped in cylinders. We capture total logical I/O usage, the number of physical I/Os and total physical I/O usage to determine the total I/O usage count per query.

5.3 AMP CPU Time

The DBQ Log Table provides logs for both normalized and raw CPU measures. Normalized CPU time is derived by applying a CPU scaling factor to the node-level raw CPU time. This CPU time is affected by the processing time, wait time and run delays observed for the AMP, the parser, and other kernel units. Adding all these features to the training set help us estimate the overall CPU processing time.

6. REGRESSION MODELS

We focus on the problem of resource prediction and evaluate various regression models to help improve predictions about the performance of an OLAP workload. Regression analysis is used for prediction and forecasting in the field of machine learning. It is utilized to understand which independent variables are related to the dependent variable, and to explore the forms of these relationships. The performance of regression analysis methods in practice depends on the form of the data generating process, and how it relates to the regression approach being used. Since the true form of the data-generating process is generally not known, regression analysis often depends to some extent on making assumptions about this process.

6.1 Ordinary Least Square Regression

With an initial assumption that the input features obtained from different sources are independent, we evaluated the performance of the Ordinary Least Square model and its accuracy in estimating the query response time, Total I/O Count and the AMP CPU time. Regression is one of the main supervised learning problem besides classification where the goal is to predict Y from X from a jointly distributed variable (X,Y) using a function:

$$f : R^d -> R$$

. Mathematically, it solves a problem of the form:

$$\min_{\omega} ||X\omega - y||^2$$

We created a regression model from the various features obtained from the DBQ Log Table and XMLPLAN table and used multiple regression models to evaluate and predict which one of the independent variables are good predictors for the dependent variable. To avoid issues of incommensurability that arise from discrepancies in units across features, we also decided to include the normalized version of the Ordinary Least Square model.

6.2 Ridge Regression

Multicollinearity is the existence of near-linear relationships among the independent variables. This creates inaccurate estimates of the regression coefficients, inflating the standard errors of the regression coefficients. We have an overdefined model created from the training set where the dimension of each row in the training set is greater than observations, leading to multicollinearity which we assumed to be the cause of high error rates for Total I/O Count and and almost zero percent error for AMP CPU Processing Time. To find a solution to this, we used Ridge regression, a technique for analyzing multiple regression data that suffer from multicollinearity. When multicollinearity occurs, Least Square estimates are unbiased, but their variances are large so they may be far from the true value. By adding a degree of bias to the regression estimates, Ridge regression reduces the standard errors and we obtain a more reliable model for resource prediction. Ridge regression uses ridge coefficients to minimize a penalized residual sum of squares, in the form given by:

$$\min_{\omega} ||X\omega - y||^2 + \alpha||\omega||^2$$

6.3 LASSO Regression

The LASSO (Least Absolute Shrinkage and Selection Operator) is a regression method that involves penalizing the absolute size of the regression coefficients. By penalizing we create a new model using the existing training set where some of the parameter estimates may be exactly zero. The larger the penalty applied, the further estimates are shrunk towards zero. The objective function to minimize is:

$$\min_{\omega} \frac{1}{2n_s}||X\omega - y||^2 + \alpha||\omega||$$

One drawback for Ridge regression is that it cannot zero out coefficients. Therefore, either all the coefficients in the model are included or none of them. In contrast, the LASSO does both parameter shrinkage and variable selection automatically. Therefore, LASSO can help us determine which features are best correlated to the output parameters.

7. EXPERIMENT

The TPC-H is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance. This benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions. Due to resource constraints, we were forced to use a scale factor of only 1 GB and we were unable to compile two of the queries provided by TPC-H, leaving 20 queries for our analysis.

We used an Asus Zenbook Prime with a dual-core Intel i5 processor clocked at 1.70 GHz. 4 GB of RAM was installed and Teradata Express for VMWare player was used as the database server. We wrote a tool called TeradataPy that processes the training sets and test sets, extracting data from the log tables using autogenerated BTEQ scripts. The regression model code makes heavy use of the Python numpy and sklearn libraries.

To evaluate the regression models, we created three different runs where our program would randomly select 14 TPC-H queries, process them as the training set, and calculate the error percentage between the predicted values for the 6 remaining queries in the test set and the actual values observed in the DBQ Log Table. Our results are shown below.

7.1 First Run

There was not much variation in terms of prediction accuracy while measuring the total response time. While predictions were erratic for total I/O count, it was still reduced considerably using the LASSO regression model, though it increased the error estimate of the test set predictions.

Figure 3: For the first run, prediction was relatively accurate for query response time.

7.2 Second Run

Another random subset of the TPC-H queries was assigned as the training set and fit into the regression model to test if the models performed similarly on the new test set. The query response time accuracy was relatively similar. However, when looking at the prediction of total I/O count, none of the models performed well in their predictions.

7.3 Third Run

Finally, looking at the third random run, we realized that the Least Square regression with normalization did not work well for predicting AMP CPU time in comparison to other models. We noted that this was the only one of the three runs where the sole TPC-H query involving creating and dropping a view was included in the training set, which may have contributed to the massively skewed predictions.

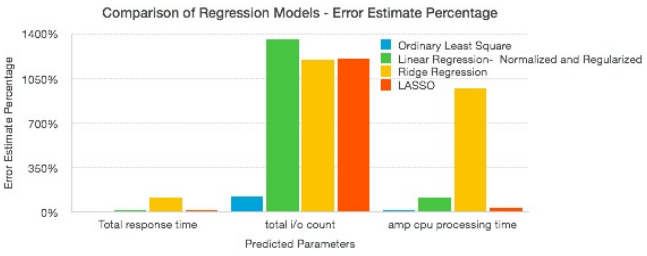

Figure 4: In the second run, only Ridge regression failed to predict AMP CPU time accurately.

Figure 5: Predictive accuracy may have been poor in the third run due to the anomalous query in the training set.

8. EVALUATION

We found that the regression model performance is highly sensitive to the input training set. There was not much variation in the TPC-H workload in terms of query response time, which is why all the models predicted response times for queries in the test set relatively well. However, for total I/O count, there were different orders of magnitude in the training set for all three runs, so the models did not get a sense of what a good estimate would be based on the features we provided, especially given such a small training set. Since the total I/O count includes logical I/Os as well, it may have been more fruitful for us to examine aggregate I/O in kilobytes, which is also recorded in the DBQ Log Table. While AMP CPU time did not have much variability, factors such as other programs context switching on the local cores could affect it.

The constrained nature of our experiment setup had a significant impact on our results. The test machine had only two cores, so only two AMPs executed the query plan for both the training and test sets. Therefore, we were not able to study the effects of parallelism on a large scale for ad-hoc query processing. The database was also stored on the local disk, so we were forced to use the smallest scale factor of 1 GB. Increasing the dataset size would lead to less overfitting and also give us a better understanding of regression model performance at a scale closer to commercial database levels.

9. CONCLUSION

We had the least estimate error measurement for the total response time of a query using the Ordinary Least Square (as well as the normalized version), although the error estimate of the total I/O count and AMP CPU time remained high in each and every model. This may be due to one of two factors - either we inaccurately determined the set of features affecting the total I/O Count and AMP CPU process time or there was a lack of sensitivity in the training set for these predictions.

The major challenge in prediction of resource utilizations is the selection of features that affect the predictions. In order to deal with the disadvantages of limited dataset, we tried to find a solution to multicollinearity using Ridge regression and LASSO. Usually, LASSO is preferred over Ridge regression when the solution is believed to have sparse features because L1 regularization promotes sparsity while L2 regularization does not. However, LASSO and Ridge regression gave erratic result in our case. Our next approach would be to use Elastic Net over LASSO because it can deal with situations where the number of features is greater than the number of samples, and with correlated features, where LASSO is most erratic.

There are many avenues for future work in this area. One main addition would be to create a custom query parser that would extract features directly from the query text. This would allow the QCD XML table to be used so that queries need not be run to obtain the features. Also, a workload with a greater number of diverse queries would help train the model better, especially if the selectivities, indexes used, and join combinations had greater variety. With these considerations in mind, we hope our work provides a framework for performing analysis on the Teradata platform.

10. RELATED WORK

Statistical machine learning has been widely used by the high-performance computing (HPC) community in the past to address performance optimization for simpler problems: Brewer [Brewer, 1995] used linear regression over three parameters to select the best data partitioning scheme for parallelization; Vuduc [Vuduc, 2003] used support vector machines to select the best of three optimization algorithms for a given problem size; and Cavazos and his colleagues [Cavazos et al., 2007] used a logistic regression model to predict the optimal set of compiler flags. All three examples carve out a small subset of the overall tuning space, leaving much of it unexplored. In addition, the above research was conducted prior to the multicore revolution, thus ignoring metrics of merit like energy efficiency. Recently, Liao and others [Liao et al., 2009] used a variety of machine learning techniques to tune processor prefetch configurations for datacenters.

11. REFERENCES

[1] Teradata database documentation. info.teradata.com.

[2] Teradata database - sql reference: Functions and operators release 12.0, 2010.

[3] A. Ganapathi. *Predicting and optimizing system utilization and performance via statistical machine learning.* PhD thesis, University of California at Berkeley, 2009.

[4] e. a. Mozafari, Barzan. Performance and resource modeling in highly-concurrent oltp workloads. In *Proceedings of the 2013 international conference on Management of data.* ACM, 2013.

APPENDIX

A. FEATURES

ampcpurundelay: The total amount of time the request was delayed in the run queue on all active AMPs before being processed.

iokb: The aggregate disk I/O is kilobytes.

minampio: The I/O count of the lowest I/O utilized AMP in the query.

maxampcpurundelay: The highest amount of time the request was delayed in the run queue on an AMP before being processed.

parsercputime: The total parser and dispatcher CPU time in seconds (with .01 resolution) used for the query.

maxampotherwaittime: The highest amount of time spent waiting for the results of another AMP.

peotherwaittime: The highest amount of time the PE spent waiting for other resources.

reqphysiokb: The required disk I/O in kilobytes.

cputime: The total CPU time in seconds (with .01 resolution) used on an AMP (or all AMPs if SUM aggregate is specified) for a query.

ampotherwaittime: The amount of time spent waiting for the results of another AMP.

pecpurundelay: The total amount of time the request was delayed in the PE run queue before being processed.

parsercpukerneltime: The kernel time spent by the parsing engine.

indextype: The type of an index.

maxampcputime: The CPU time in seconds (with .01 resolution) of the AMP with the highest CPU utilization in the query

peiowaittime: The total amount of wait time caused by I/O rate handling for the PE during the life of the request.

B. OUTPUT PARAMETERS

Query Response Time: This parameter is calculated as firstresptime - starttime.

totaliocount: Total I/O used by the query.

ampcputime: Total AMP time used for the query in CPU seconds.