Machine Learning for Automatic Physical DBMS Tuning

Ivan Beschastnikh and Andrew Guillory

December 6, 2007

Abstract

Tuning a DBMS that experiences varying workload is challenging. Database administrators cannot be expected to monitor the workload and react with appropriate tunings, therefore automation is essential. In this report we outline a new method for automatic physical DBMS tuning that uses machine learning to model and predict workloads, and tune for the future. Our method builds on previous approaches by exploiting predictability of DBMS workload without assuming full knowledge. We present results on a set of synthetic workloads generated from the TPCH query templates.

Figure 1: YouTube.com is an example of a high-traffic site that exhibits predictable time-dependent visitation patterns. Learning such workloads and tuning at appropriate times may improve throughput.

1 Introduction

The performance of a DBMS is highly dependent on the physical design of the database. Automatic physical design tuning seeks to automatically choose a physical design based on workload information. Previous approaches to this problem have been limited by assumptions concerning knowledge of the workload. We propose to use machine learning (ML) to overcome these limitations by learning a model of the history of the workload and predicting changes in the workload.

Application that benefit from automatic DBMS tuning include data warehousing architectures, popular high-traffic web-sites, and many more. For example, consider a data warehousing application that collects sales information from all the regional stores on a daily basis during the weekdays. After receiving a week of data, the application proceeds to compute over the data on the weekends. This scenario is meant to illustrate that time varying workload may be stable and predictable, allowing for time-dependent tuning of the DBMS. Throughout the paper, we motivate our parts of our approach by using an example DBMS backend designed for a retail store that experiences different workloads on weekdays than on weekends.

Our second example application is YouTube.com that attracts millions of users daily. The visitation workload of YouTube for the recent month is plotted in Figure 1. The peak visit times occur on the weekends. Learning this pattern and using it to optimize the underlying DBMS may benefit user throughput, and page-load latency. For example, the DBMS may be tuned for high traffic on the weekends and weekdays after 6pm. However, on weekday mornings, the DBMS may be tuned to support data mining activities.

In the next section we review related work on automatic tuning. We then present a step by step overview of our approach in Section 3. We explain our experimental methodology and present experimental results in Section 4 and Section 5. We then proceed with discussion and future work in Section 6. We conclude with Section 7.

2 Related Work

Offline Set Based Tuning. Early work on automatic physical design tuning focused on recommending indexes by considering example database workloads. The example workloads usually take the form of a set of typical queries to the database. This work has since been integrated into several major commercial database management systems (e.g. MS SQL Tuning Advisor) and is a significant step towards automating database tuning. A drawback of set-based tuning is that it requires a database analyst to decide on a single representative workload in advance. Real world database applications often involve temporally varying workloads which may, as previously discussed, dramatically change over time

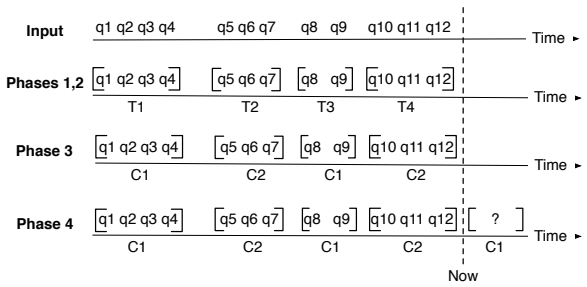

Figure 2: ML phases in our algorithm. The q_i are queries, T_j are tunings associated with [bracketed] sets of queries, and C_k are clusters of tunings learned (and then predicted) by the algorithm.

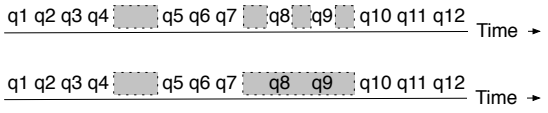

Figure 3: Query workload can be segmented so that times between query groups are long enough to allow for DBMS re-tuning (top). If such a segmentation does not exists, an algorithm with a threshold for the maximum number of queries allowed in a gap can be used (bottom).

Offline Sequence Based Tuning. Follow up work attempted to overcome the limitations of set based tuning by using a sequence of queries [1]. This work allowed for temporally varying workloads unlike previous set-based approaches. However, this approach again requires the assumption that a database analyst knows the workload – the set of queries and their ordering.

Online Tuning. Most recent approaches automatic tuning as an online problem [2]. In this approach new tunings are recommended automatically as the workload changes. Like the offline sequence based approach, this approach can adapt to changes in the workload. However, unlike the offline approaches, the online model requires no up-front knowledge concerning the sequence or the actual queries composing the workload. One problem with the online method presented in [2], is that it does not make any attempt to predict changes in the workload – it reacts to changes as they occur. Moreover, as the physical tuning requires substantial compute time, tunings typically lag behind workload changes. As a consequence of this lag, the algorithm in [2] requires non-trivial modifications to avoid tuning oscillation.

3 Proposed Approach

We suggest an approach to combine the strengths of the offline sequence based approach and the online approach. Our tuning advisor executes online as in [2], while maintaining a temporal model of the workload with it can use to predict and plan ahead for workload changes as in [1]. Here we present our prediction model of database workloads. We also present a simple decision algorithm to interpret predictions, and to decide on the best course of action.

Our method takes in a historical trace SQL queries. It groups these queries based on their co-occurrence into sets, for which it then generates tunings using the MS SQL tuning advisor. It then clusters tunings based on tuning features. Next, it outputs a probability distribution over clusters. This distribution encodes confidence of the system as to the most likely sequence of tuning clusters to occur next. These predictions are then used to decide what changes if any to make to the physical design of the DBMS. These phases are illustrated in Figure 2).

Ideally the above process is repeated as new queries arrive. To avoid overhead, the most expensive steps (running of the MS SQL tuning advisor in particular) can be run less frequently. In our experiments, we *train* by executing the learning steps once at the beginning of the experiment.

3.1 Trace Segmentation

In the first phase we segment the query workload into temporally coherent groups of queries. The purpose of this phase is to identify periods of time in the workload during which we could have made changes to the database tuning (alternatively identify times during which we could not have made tuning changes). The assumption here is that it is unacceptable to make changes to the database tuning during periods of high workload. This assumption may be easily relaxed depending on the application. For example, a database backend for a retail store may experience many queries during the store operating hours when purchases are being made and fewer queries at night. In this phase we want to identify the store operating hours so that we can avoid making changes to the tuning during these times.

There are a number of methods which could be used to segment the trace. One simple approach would be to look for gaps in between queries longer than a threshold time value set to the minimum time it takes to re-tune the database. A problem with this approach is that there might be workloads for which there are no idle time periods in the workload that are large enough to admit a retuning. For this case we can redefine our threshold based on query frequency within a sliding window of time or use a more advanced technique to segment the sequence such as normalized cuts segmentation. Figure 3 shows these two ap-

proaches. The shaded areas are pauses during which re-tuning is possible. In our experiments we use the simple gap threshold with an additional heuristic which allows for segments with less than 6 queries.

3.2 Segment Tuning

In the second phase of the ML algorithm, we generate tunings for each of the segments of queries from step one. These tunings are sequences of CREATE INDEX and CREATE STATISTICS statements that the tuning advisor generates for the set of SQL queries for each of the segments from step one of the algorithm. For our retail store example, we would generate tunings for every day of the week for which we have identified the working hours in phase one.

Note that we have essentially computed an offline tuning sequence for the past sequence of queries. We could, for example, use the more complicated offline sequence based tuning method in [1]. This method simultaneously segments and computes tunings for a sequence of queries taking into consideration the cost of changing tunings. As we do not have an implementation of this method, we instead use a simpler approach to take advantage of an offline set based tuner – the SQL Server tuning advisor. In the next two phases we essentially extrapolate this offline tuning to future dates.

3.3 Tuning Clustering

In the this phase we cluster the tunings from the prior step based on the tables and fields that correspond to the indices and statistics of the tunings. The tuning clusters correspond to similarly performing tunings. This clustering phase is necessary as segments of queries with identically distributed queries may produce slightly different tunings because of small differences in the frequencies of queries or even randomization in the tuning advisor. In our running example, the retail store may experience different purchasing patterns on weekdays and weekends. This could then cause the weekday tunings to be significantly different from weekend tunings. In this phase, we would group together the weekday and weekend tunings into two distinct clusters.

We use a spectral clustering algorithm based on [4]. Our clustering algorithm takes in as input a similarity matrix M where $M_{i,j}$ is a measure of the similarity between tuning i and tuning j. To compute the similarity between two tunings, we first count the number of shared indices, ignoring columns referenced in INCLUDE statements. [1] We then use the exponential of this count as a similarity measure.

[1]This procedure can be interpreted as a dot product of two large sparse binary vectors. However, we do not explicitly compute these vectors.

To produce clusters from the similarity matrix, we replace the diagonal entries of the matrix (self similarity values) with 0, this step was normalizes for the differences in the numbers of indices between tunings. We then normalize the rows of the matrix to sum to 1 to create a stochastic matrix. As described in [4], the first eigenvector (eigenvector corresponding to the largest eigenvalue) of this matrix is normally constant, and the second eigenvector of the matrix gives a relaxed solution to the normalized cuts clustering problem. When this solution is tight, the sorted values of the second eigenvector are piecewise constant. To group the tunings into k clusters, we run k-means on the rows of the matrix $v_2, v_3, ...v_k$ where v_i is the eigenvector corresponding to the ith largest eigenvalue.

3.4 Prediction

In this step, we train a classifier to predict future sequences of clusters. We create an additional cluster class composed of gaps in the workload so that the classifier can help predict these breaks. Our classifier takes as input the prior sequence of clusters and features associated with these clusters such as time of day, day of the week, etc. The classifier then outputs cluster predictions for times in the future. Back to our store example, the classifier would learn that weekdays during working hours correspond to the first cluster and that weekends during working hours correspond to the second cluster. For future dates, the classifier would predict this same pattern.

In our system, we use logistic regression [3] because it is reasonably fast and produces probabilities over clusters as output. We train a separate binary classifier for each cluster and for gaps between clusters (one against all multiclass classification). Each classifier takes in as input a sparse binary vector encoding time of day and day of week and produces as output a probability. As the classifiers are trained separately for each cluster, these probabilities do not sum to 1 across clusters. We normalize these probabilities before proceeding to the next step. As an example, Figure 5 plots cluster probabilities over time for one of our experiments.

3.5 Decision Making

Having produced predictions of clusters for future dates. The system must decide as to what course of action to take next. There are a number of decisions that could be made. The system could decide that it is not in its best interest to re-tune the DBMS. This can happen if the predicted tunings improve DBMS performance for only a few, or many insignificant queries in the workload. This can also happen if the gap during which the DBMS will be tuning is predicted to conflict with the workload, thus increasing the

execution time for queries that execute concurrently with the tuning process.

The system could also decide that it needs to do the re-tuning – it might be apparent that the coming gap will not include any workload with high probability, indicating that this time could be used for tuning the DBMS if an appropriate tuning is available. The question then is what tuning should be uses. One approach is to use the most likely tuning that is predicted in the near future. However, the length of the tuning cluster may impact this decision. For example, it makes more sense to tune for a cluster that will dominate the probability space during the next 3 or more hours rather than optimize for the next cluster as the next cluster might persist only for a short time.

In our experiments we re-tune the database whenever the classifier predicts a gap and the following predicted cluster is different from the currently tuned for cluster. When tuning for a particular cluster, we use tuning advisor recommended tunings created for the union of all the queries that fall into the cluster in the training workload.

4 Test Methodology

We have created a 1 GB instance of the TPCH benchmark database using MS SQL Server 2005 and written code to generate workload sequences using the TPCH query templates. We use all of the TPCH query templates except for one which creates and then drops a temporary table which we had difficulty executing with SQL Server. These workloads are meant to roughly reflect a database workload that we imagine a real DBMS might experience. We have also written code to compare tunings for sets of queries and to extract features from tuning advisor recommendations.

Our first test workload, called $W_{selects}$, consists entirely of SELECT statements. For this workload we divided the query templates from the TPCH benchmark into two subsets: Q_{fast} and Q_{slow}, consisting of queries which take less than 10 seconds and more than 10 seconds to complete on an untuned database respectively. In our workload, queries are chosen uniformly at random from Q_{fast} except for Sundays for which queries are chosen uniformly at random from Q_{slow}. Between 9 AM and 5 PM, queries occur each minute with probability .25, and during other times queries occur with probability .005. For this workload we use a month of training workload, with which we tune and build a predictive model, and a week of test workload on which we test our model.

We implemented our system in Python, interfacing with SQL Server through pyodbc and the sqlcmd. We use our own implementation of spectral clustering using numpy and the implementation provided by the authors of [3] for binary logistic regression. We compare our method to offline set based tuning (e.g. tuning on all queries in the training workload as one cluster).

For our experiments we use SQL Server Database Tuning Advisor. We restrict the tuning advisor to run for 30 minutes and also randomly sub-sample to ensure that workloads given to the tuning advisor consist of no more than 1000 queries. We have found that with too many queries, the tuning advisor is unable to make recommendations in the allotted time.

5 Results

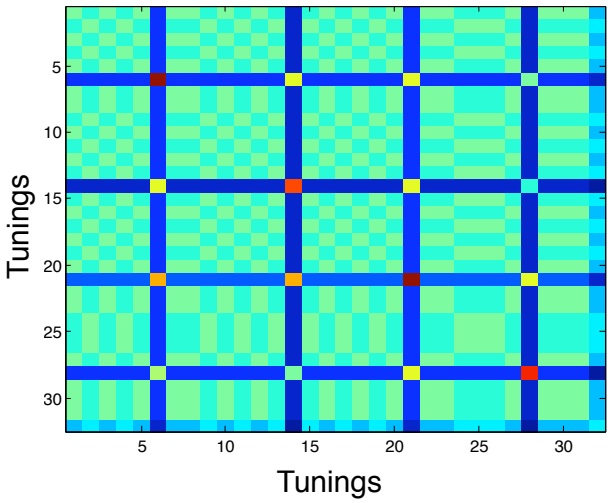

Figure 4: A similarity matrix of tunings for different segments in $W_{selects}$ workload. Darker colors indicate higher similarity. The dark bands represent weekend tunings, the light bands represent weekday tunings.

Our simple segmentation strategy is sufficient separate out the 9 AM to 5 PM segments in $W_{selects}$, with one additional outlier segment incorrectly identified during off hours. Figure 4 shows the tuning similarity matrix for these segments. The dark bands in the similarity matrix correspond to Sundays, the light bands correspond to weekdays. Our clustering method correctly identified these two groups. Figure 5 shows cluster probabilities over time. The red cluster corresponds to weekends, the blue corresponds to Sundays. The probabilities correctly correlate with the workload for these days.

The total running time for $W_{selects}$ using our approach and using the set-based approach for comparison are plotted in 6. As visible from the figure, our approach performs worse than the set-based approach on both clusters. We believe that this is because of limitations of using the tuning advisor for our recommendations. We discuss these limitations in the next section.

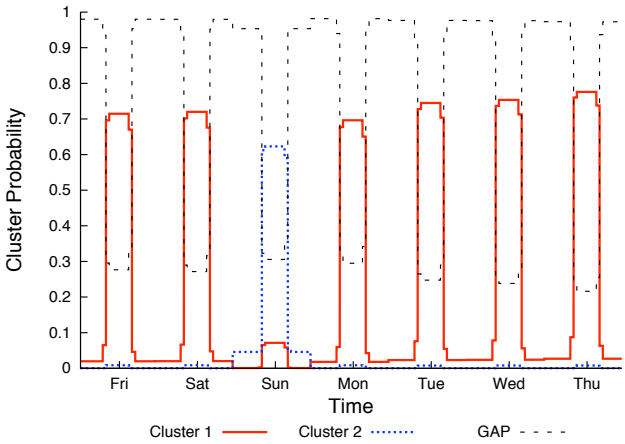

Figure 5: Tuning cluster probabilities for three clusters of the W_{select} workload over time. Cluster 1 corresponds to weekdays, and cluster 2 corresponds to weekend. The GAP cluster corresponds to times of day for which no queries were run. The probabilities are normalized to sum to 1.

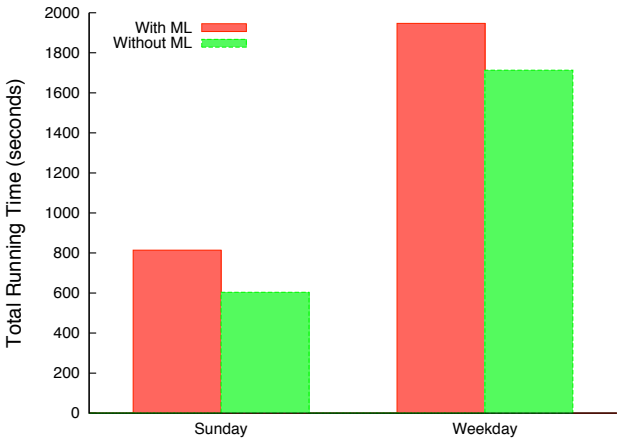

Figure 6: Total execution time comparisons for the ML approach and the non-ML approach across the two query clusters in our $W_{selects}$ test workload. Our system performs slightly worse than the set-based tuning approach.

6 Discussion

Necessary workload properties Our approach makes certain assumptions concerning properties of the database workload. We think these assumptions are reasonable, but we haven't validated them on real world workloads.

In the segmentation phase of our approach, we assume there are identifiable regions within the workload trace during which we can tune and that these gaps separate segment the workload. In the clustering phase, we assume that different workloads will produce different database tunings. Finally, in the prediction phase, we assume that the sequence of workload changes is predictable and periodic.

Necessary tuner properties Even if our ML algorithm were to identify and predict the tuning clusters exactly, our approach would not benefit the DBMS unless the tuning advisor used has a particular property. We believe that this constraint is fundamental in order for the tunings recommended by the tuning advisor to be useful to any prediction algorithm, including our ML algorithm. By proxy, this property is also necessary for our ML algorithm to be relevant to a DBMS.

Define two arbitrary sets of queries Q_1, Q_2 and two tunings T_1, T_2 that are generated by the tuning advisor for each of the query sets respectively. The property we need states that the *average* performance of the database on queries in Q_1 tuned with the set of tunings in T_1 should be no worse than the performance on queries in Q_1 tuned with tunings in T_2. Ideally, the average performance of T_1 should be significantly better than T_2 on Q_1 (assuming that Q_1 and Q_2 are different – otherwise there would be no reason to change the database tuning).

We've found that the SQL Server Tuning advisor only sometimes satisfies this property. We believe that this is the reason for the performance illustrated in Figure 6.

6.1 Future Work

Besides testing on real world workloads, there are many opportunities for improving our approach. An important improvement to the decision making algorithm involves reacting to workloads that deviate from what has been predicted. If there is a deviation, the decision engine might need to re-evaluate whether tuning the DBMS is appropriate (perhaps the tuner should stop immediately if it is in the middle of a tuning). More work may also use observed workload to offset predictions and to learn new variations and generate new cluster probabilities as the observed workload varies and deviates from the training data over time.

For our system, we selected the parameters of our segmentation and clustering methods by hand (in particular the thresholds for our segmentation method and the number of clusters for the clustering method). To perform well on real world data sets we need methods for determining these parameters automatically. It would be interesting to experiment with simultaneously segmenting and clustering the workloads.

Our initial YouTube example also demonstrates that learning workload patterns may be useful for scheduling DBMS applications. In this scenario, non-critical tasks can be automatically scheduled to execute during times of low visit activity. As these patterns may change over time, an

automatic learning-based approach is of high relevance to the problem of scheduling applications that use the DBMS.

7 Conclusion

We have created a system to automatically tune a DBMS for time-varying, but predictable workloads. Our method segments the workload, tunes on the segments, clusters the tunings, and uses logistic regression to predict future tuning clusters. Our results indicate that the approach shows promise and we think it may be improved dramatically. We have also found that certain properties of the recommendations made by the tuning advisor must hold in order for our algorithm to successfully improve performance. We have outlined the steps of our approach and discussed issues and future work relevant to applying machine learning to DBMS tuning.

References

[1] S. Agrawal, E. Chu, and V. Narasayya. Automatic physical design tuning: workload as a sequence. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 683–694, New York, NY, USA, 2006. ACM Press.

[2] N. Bruno and S. Chaudhuri. An online approach to physical design tuning. In *ICDE '07: Proceedings of the 23rd International Conference on Data Engineering*, pages 826–835, 2007.

[3] P. Komarek and A. Moore. Making logistic regression a core data mining tool: A practical investigation of accuracy, speed, and simplicity. Technical Report CMU-RI-TR-05-27, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2005.

[4] M. Meila and J. Shi. A random walks view of spectral segmentation. In *AI and Statistics (AISTATS) 2001*, 2001.