ISSN-2230-9659

Study of Different Intelligence Level of Workload within Completion Times

Nirbhai Singh
Research Scholar, Manav Bharti University, H.P., INDIA

IGNITED MINDS Journals

ABSTRACT:-

While planning the execution of report-generation workloads, database administrators often need to know how long different query workloads will take to run. Database systems run mixes of multiple queries of different types concurrently. Hence, estimating the completion time of a query workload requires reasoning about query mixes and inter-query interactions in the mixes; rather than considering queries or query types in isolation. This paper presents a novel approach for estimating workload completion time based on experiment-driven modeling and simulation of the impact of inter-query interactions. A preliminary evaluation of this approach with TPC-H queries on IBM DB2 shows how our approach can consistently predict workload completion times with good accuracy.

I. INTRODUCTION

Data warehouses, and Business Intelligence (BI) workloads that run on these warehouses, are an important and growing segment of the database market [1]. Many BI workloads are long-running batch workloads that get executed repeatedly at different periods. An important question to ask about a batch BI workload is: “How long will this workload take to complete?” The answer to this question is useful in many workload management contexts. For example, this question arises when a database administrator (DBA) is deciding whether the execution of a report-generation workload will fit within the available batch window. A tool that estimates the completion time of a BI workload can also be used as a what-if module. For example, the DBA can consider different ways to reorder the workload or partition the workload in a parallel system, and ask how long each execution would take.

Unfortunately, the state of the art does not provide a database administrator with any tools that predict the completion time of a batch BI workload. In this paper, we address this problem and propose an approach for predicting the completion times of such workloads. A unique and defining feature of our approach is that it takes query interactions into account. At any point in the execution of a typical workload in a database system, the system will be running a mix of queries of different types. These queries run concurrently and interact with each other, and this interaction can have a significant impact on performance. Sometimes this impact can be positive and sometimes it can be negative. For example, a query Q_1 can bring data into the buffer pool that is then used by a concurrently running query Q_2 (an example of positive interaction). Alternatively, Q_1 and Q_2 could interfere with each other on hardware resources such as CPU or memory, or on internal database system resources such as latches or locks (all examples of negative interaction).

In order to demonstrate the significant impact of query interactions, we use queries from the TPC-H decision support benchmark with a database size of 10GB running on DB2 (our experimental setting is described in Section III). Table I shows the run time of the 6 longest running TPC-H queries when they run alone in the system, which we denote by tj .

IGNITED MINDS Journals

Table II shows three query mixes for this setting. For each mix, the table shows the number of queries of each type, N_{ij} , and the average run time of each query type, A_{ij} . The high variability in Aij illustrates the impact of query interactions. For example, consider the performance of Q_7 in the two mixes m1 and m3. Mix m1 presents an example of positive interaction for Q_7. The average run time of Q_7 in this mix is 72:7 seconds, while the run time of Q_7 when it is run alone in the system is 102:06 seconds. Thus, Q_7 benefits from being run in this mix. On the other hand, Q_7 suffers due to negative interaction in mix m3. Mix m2 presents another example of positive interaction, this time for Q_{18}.

We emphasize that these positive interactions are not due to the simple benefit of concurrent execution where individual query run times increase when run together, but the overall completion time is less than the time required to run the queries one at a time. Instead, we see here that every instance of Q_7 (or Q_{18}) takes less time in mix m1 (or m2) than when it runs alone. Further demonstration of the impact of query interactions in query mixes can be found in [2].

Figure 1 illustrates how much the interactions in query mixes can impact the end-to-end run times of different workloads. The figure shows the run times of two workloads. Both workloads consist of exactly the same 60 instances of TPC-H queries running on a 10GB database on DB2. The database physical design and the tuning parameters of DB2 are the same for both workloads. The only difference between the two workloads is the arrival order of the queries, which results in different query mixes being executed by the system. This simple change results in the completion time varying from 3:3 hours to 5:4 hours. In Workload 1, queries that compete for resources get executed concurrently, resulting in negative interactions. In Workload 2, queries that help each other get executed together, resulting in positive interactions. The 2.1 hour difference in performance is completely attributable to different query interactions in the different runs. Figure 1 also shows the completion time predictions of our interaction-aware solution, and it is clear that these predictions are quite accurate.

Query Type	$Q1$	$Q7$	$Q9$	$Q13$	$Q18$	$Q21$
Run Time t_j (sec)	294.61	102.06	578.61	101.27	554.56	570.37

TABLE I : RUN TIME tj (IN SECONDS) OF DIFFERENT TPC-H QUERY TYPES ON A 10GB DATABASE

Mix	$Q1$		$Q7$		$Q9$		$Q13$		$Q18$		$Q21$	
	N_{ij}	A_{ij}	N_{ij}	A_{ij}	N_{ij}	A_{ij}	N_{ij}	A_{ij}	N_{ij}	A_{ij}	N_{ij}	A_{ij}
m_1	1	1897.4	2	72.7	5	2919.3	0	0.0	2	1904.1	0	0.0
m_2	4	538.0	0	0.0	0	0.0	0	0.0	1	539.3	0	0.0
m_3	0	0.0	4	264.5	0	0.0	0	0.0	1	3413.7	0	0.0

TABLE II : AVERAGE RUN TIME A_{ij} (IN SECONDS) OF DIFFERENT QUERY TYPES IN QUERY MIXES ON A 10GB DATABASE

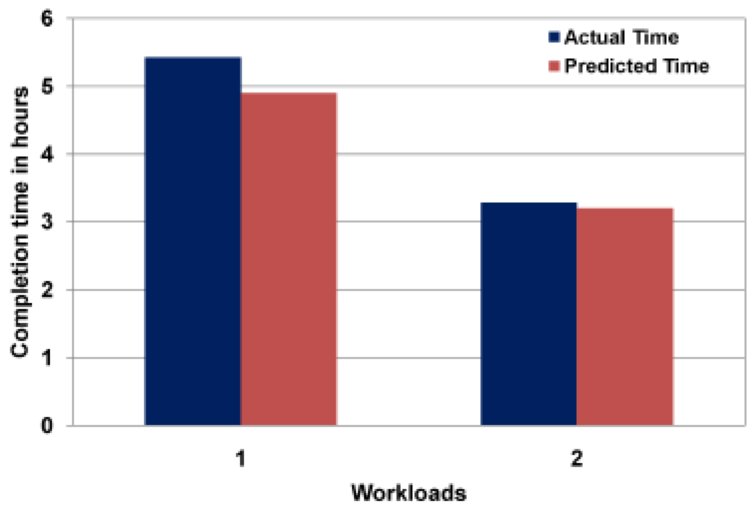

Fig. 1. Workload completion time for different arrival orders

We are not aware of any work focusing on predicting the completion time of BI workloads, particularly in an interaction-aware manner. Overall, there is very little work that deals in a general way with the performance of concurrently executing query mixes and the interactions within these mixes.

In our prior work ([3], [4]), we have addressed the issue of interaction-aware query scheduling and presented solutions that significantly improve performance over interactionoblivious schedulers.

In this paper, as well as in [3] and [4], we use experiment-driven performance modeling to capture the effect of query interactions.

Experiment-driven performance modeling is gaining wide acceptance as a way to build robust performance models for complex systems. A relevant work from this area is [5], which uses statistical learning techniques to predict performance metrics for database queries. That paper is able to make performance predictions for individual query types with less than 20% error for 85% of the test cases. However, the paper focuses exclusively on single query types and does not consider interactions and query mixes, which are our focus in this paper. By using our interaction-aware techniques, we are able to achieve prediction accuracy similar to [5] for batch BI workloads with interacting queries.

We present our approach for predicting the completion time of a workload in Section II. Section III presents a brief empirical evaluation of this approach using TPC-H queries on DB2. We conclude in Section IV.

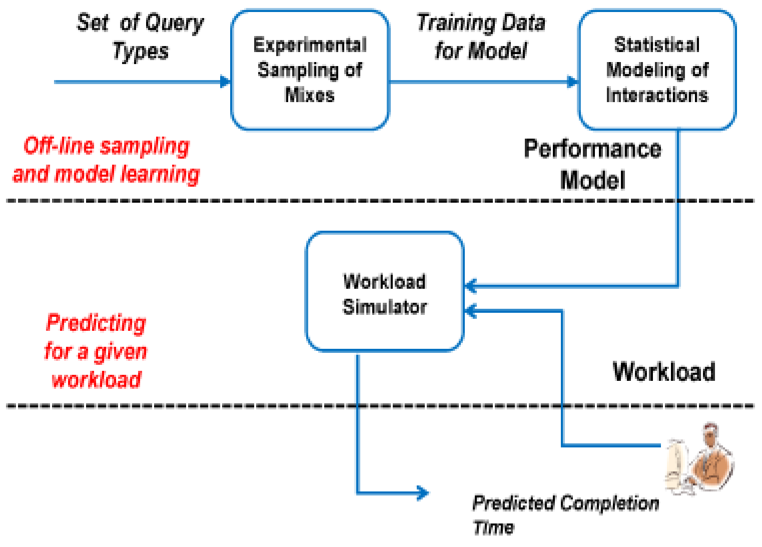
Set of Query Types
Experimental Sampling of Mixes
Training Data for Model
Statistical Modeling of Interactions
Off-line sampling and model learning
Performance Model
Workload Simulator
Predicting for a given workload
Workload
Predicted Completion Time

Fig. 2. Solution overview

II. PREDICTING COMPLETION TIME OF A WORKLOAD

An overview of our solution is presented in Figure 2. Our solution has two parts: (1) an experiment-driven model learning component that we use to build interaction-aware performance models, and (2) a workload simulator that uses these performance models to predict the completion time of a given workload. We assume that the set of query types is known a priori, determined by the DBA. When predicting the execution time of a given workload, we assume that the full batch of queries in this workload is known and queued for execution. The queries are dispatched to the database system and they execute concurrently until the workload completes.

We assume that the number of queries that execute concurrently, also known as the multi-programming level (MPL), is fixed, which is typically the case in batch BI systems [6]. Next, we describe the two parts that make up our solution.

Experiment-driven Modeling: To predict the completion times of different workloads, we need interaction-aware performance models that predict the completion times of individual query types in different query mixes. It may be possible to observe different query interactions through passively monitoring the workloads in a production system. If we monitor the execution of production workloads, we could determine which query mixes are actually encountered in these workloads, how long each mix runs, and what effect each mix has on the completion time of each query type. We could then train statistical models for the performance of different query types based on these observations. This, however, cannot guarantee comprehensive coverage of the space of possible query mixes and can therefore result in inaccurate models. Thus, there is a need to generate a representative set of sample query mixes and to train the models based on these samples.

Our approach to building performance models is to run experiments to collect samples from the space of possible query mixes and fit statistical models to the observed query performance in these samples. This experiment-driven modeling is an off-line process that is done once for a given set of query types. The models generated via this one-time process can be used to predict the completion time of any future workload composed of queries from this set of query types.

The model for a given query type, say Q_j, is trained from a set of n samples, where sample $s_i, 1 \leq i \leq n$,has the form $s_i = \langle m_i, A_{ij} \rangle = \langle N_{i1}, \ldots, N_{iT}, A_{ij} \rangle$. Sample si denotes an observation that the average run time of Q_j queries when run in mix mi is A_{ij} (T is the number of query types). One simple technique to generate a representative set of samples is to choose randomly from the space of possible query mixes. However, random sampling is inefficient from the modeling perspective because mixes from the same local space may be repeated unnecessarily.

The family of space-filling designs contains more efficient sampling techniques. Latin Hypercube Sampling (LHS) comes from this family and performs well in practice [7]. LHS has the nice properties of efficiency and good coverage of the mix space. It has successfully been used in other work on database systems (e.g., [8], [9]). In our setting, we adjust the mix generated by LHS such that $\sum_{j=1}^{T} N_{ij} = M$, where M is the multi-programming level of the system. Also, we observed that the number of distinct query types in a mix m has a strong impact on query interactions. Let us define the interaction level of a mix m as the number of distinct query types in m. The maximum number of interaction levels possible in the system is num ILs = min(T;M). We make sure that our set of representative mixes contain roughly equal number of samples for all interaction levels in $\{1, \ldots, num\_ILs\}$.

Sampling the space of possible query interactions is the first step towards modeling the effect of these interactions on performance. The next step is to fit a statistical model to the observed performance in the samples. Our goal is to obtain a function for each query type Q_j of the form $A_j = f(N_1, N_2 \ldots, N_T)$, where $f(\cdot)$represents the statistical model. The form of $f(\cdot)$ depends on the type of model that we use (the model structure). There are many well-known model structures, such as linear regression, regression trees, locally weighted linear regression, and Gaussian processes.

The choice of model structure impacts model accuracy, but if the training data is representative, then a good model can typically be found easily. In our work, we use Gaussian processes [10] since we found them to be a good model structure that is accurate for a broad spectrum of query mixes.

Workload Simulator: To estimate the completion time of a given workload, we use a workload simulator that simulates the changing query mixes during workload execution. To predict these changing query mixes and estimate the time that each mix will run for, the workload simulator uses the interaction-aware performance models built in our off-line modeling phase. From the run times of the mixes, the simulator estimates the completion time of the entire workload.

We consider the execution of the workload as a sequence of mixes of M queries each, where M is the multi-programming level of the system. These mixes, which we call workload phases, change when one query finishes and another starts.

The simulator tracks the fraction of total work completed by each query in each phase. Consider a query instance, qj , of type Q_j. This query instance will start with the start of some workload phase, and this workload phase would be query phase 1 for this query instance. The query will execute through different workload phases until it completes all the work it needs to perform. Let wc_{ij} be the fraction of qj 's work completed in its query phases 1 to i. When qj starts, its $wc_{ij} = 0$, and qj is done when its $wc_{ij} = 1$. We define the following recurrence relation to keep track of wcij through the different query phases:

$$\begin{aligned} wc_{0j} &= 0 \\ wc_{ij} &= wc_{(i-1)j} + (1 - wc_{(i-1)j}) * f_{ij} \qquad (1) \end{aligned}$$

The fraction of qj 's work completed up to query phase i-1 is $wc_{(i-1)j},$ and the remaining work after phase i - 1 is $(1 - wc_{(i-1)j}).$ The fraction of this remaining work that is completed during query phase i is $f_{ij} = l_i/a_{ij},$where l_i is the length of phase i and a_{ij} is the predicted remaining completion time of query instance qj when it executes in the query mix of phase i. If qj continues executing in this mix, it would finish in time a_{ij} . Since phase i will end in time li, qj will only complete l_i/a_{ij} of its remaining work in this phase.

To estimate a_{ij} , the simulator uses the performance model to obtain the estimated completion time of qj in the mix of phase i, $\hat{A}_{ij}.$ This is the time required for q_j to execute from start to finish in this mix. Since q_j has already completed $wc_{(i-1)j}$ of its work, the simulator multiplies the estimate $\hat{A}_{ij}$ by $(1 - wc_{(i-1)j}).$

$$a_{ij} = (1 - wc_{(i-1)j}) * \hat{A}_{ij} \qquad (2)$$

IGNITED MINDS Journals

To estimate l_i, the length of phase i, we observe that phase i will continue until one of the running queries finishes, at which point the simulator will transition to phase i+1. Thus, phase i will end at the earliest time a query finishes. That is,

$$l_i = \min_{j=1\ldots M} (a_{ij}) \qquad (3)$$

In phase i, the simulator uses Equation 1 to update the work completed for all queries running in this phase. After this update, some queries will have $wc_{ij} = 1,$ and these queries are finished and removed from the mix. The next queries in the workload will take their place to start phase i + 1. In phase i + 1, the query mix is different from the one in phase i, so the simulator uses Equation 2 to recompute the estimated remaining time $a_{(i+1)j}$ for all queries in the mix. The simulator then estimates the length of phase i+1 using Equation 3. The simulator then transitions from phase i+1 to phase i+2, and this continues until all $|W|$ workload queries are executed.

The simulator estimates the completion time for the whole workload, LW, as the total length of all the workload phases:

$$L_W = \sum_{i=1}^{|W|-M+1} l_i$$

III. EXPERIMENTS

Our experiments are run on a machine with dual 3.4GHz Intel Xeon CPUs and 4.0GB of RAM running Windows Server 2003. The database server we use is DB2 version 8.1. We use the TPC-H database with scale factors 1GB and 10GB. The buffer pool size of the database was set to 400MB and 2.4GB for the 1GB and 10GB databases, respectively. We use all 22 TPC-H query types except for Q_{15} which creates and drops a view. We generate different workloads for our experiments by varying the database size, the number of query types, the arrival order of the queries, the MPL, and the scheduling policy. Some workloads use all 21 TPC-H query types, while others use the 6 or 12 longest running query types. To vary the arrival order of the queries in the workloads, we generate each workload by going over the different query types in a round-robin fashion and placing B instances of each query type in the arrival queue, until all queries are in the queue. By varying B, we vary the skew in the arrival order. For some workloads, we use First Come First Served as the scheduling policy and for others we use Shortest Job First. In total, we generate 90 different workloads with actual completion times ranging from 30 minutes to more than 5 hours. Our metric for evaluating the accuracy of completion time prediction for a given workload is the relative error in predicted completion time, defined as: $rel = \frac{|pred-act|}{act} \times 100$, where pred is the predicted and act is the actual completion time. To build the performance models required by our workload simulator, we collect samples and train a Gaussian processes model using the Weka data mining toolkit [11] .

Figure 3 shows the cumulative frequency distribution of the relative error in prediction for the 90 workloads used in our experiments. (A cumulative distribution towards the upper-left corner represents lower error than one towards the lower-right corner.) The figure shows the error for two cases. In one case, we use performance models trained on 5T samples and in the other we use models trained on 10T samples (recall that T denotes the number of query types). Thus, for 21 TPC-H

query types, we collect no more than 105 sample mixes in the case of 5T, and no more than 210 sample mixes for 10T.

From Figure 3, we can see that the prediction errors in case of 5T training samples are less than 20% about 80% of the time. If the DBA has a larger sampling budget and is willing to collect up to 10T samples, then the overall accuracy improves to the point where the prediction errors are less than 20% around 90% of the time. These end-to-end results show that our sampling, modeling, and workload simulation algorithms result in accurate and robust predictions across a wide range of workloads. The DBA can now make highly

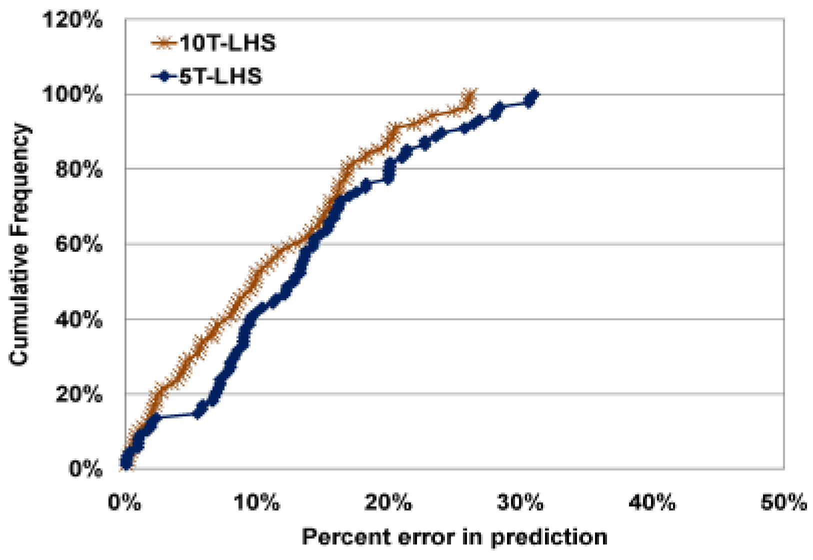

Fig. 3. Prediction error across all workload runs for 5T and 10T training Samples

accurate predictions for future workloads in her database by collecting a small number of samples just once (which can be done along with initial system setup and tuning).

IV. CONCLUSION

DBAs in a business intelligence setting often need to predict the completion time of different batch workloads. In this paper, we present an approach for predicting workload completion times that takes into account the effect of interaction among concurrently running queries. This approach relies on: (1) experiment driven performance modeling, and (2) a workload simulator that uses the performance models to simulate the execution of a workload and thereby predict its completion time.

An experimental evaluation of our approach demonstrates that it can predict completion times with a high degree of accuracy across a broad spectrum of workloads.

IGNITED MINDS Journals

REFERENCES

[1] D. Feinberg and M. A. Beyer, "Magic quadrant for data warehouse database management systems," Gartner Research Note, 2008, mediaproducts. gartner.com/reprints/microsoft/vol3/article7/article7.html.

[2] M. Ahmad, A. Aboulnaga, and S. Babu, "Query interactions in database workloads," in Proc. Int. Workshop on Testing Database Systems (DBTest), 2009.

[3] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala, "QShuffler: Getting the query mix right," in Proc. Int. Conf. on Data Engineering (ICDE), 2008.

[4] ——, "Modeling and exploiting query interactions in database systems," in Proc. ACM Conf. on Information and Knowledge Management (CIKM), 2008.

[5] A. Ganapathi, H. Kuno, U. Dayal, J. Wiener, A. Fox, M. Jordan, and D. Patterson, "Predicting multiple metrics for queries: Better decisions enabled by machine learning," in Proc. Int. Conf. on Data Engineering (ICDE), 2009.

[6] A. Mehta, C. Gupta, and U. Dayal, "BI Batch Manager: A system for managing batch workloads on enterprise data warehouses," in Proc. Int. Conf. on Extending Database Technology (EDBT), 2008.

[7] C. R. Hicks and K. V. Turner, Fundamental Concepts in the Design of Experiments. Oxford University Press, 1999.

[8] S. Duan, V. Thummala, and S. Babu, "Tuning database configuration parameters with iTuned," in Proc. Int. Conf. on Very Large Databases (VLDB), 2009.

[9] S. Tozer, T. Brecht, and A. Aboulnaga, "Q-Cop: Avoiding bad query mixes to minimize client timeouts under heavy loads," in Proc. Int. Conf. on Data Engineering (ICDE), 2010.

[10] T. J. Santner, B. J. Williams, and W. Notz, The Design and Analysis of Computer Experiments, 1st ed. Springer, 2003.

[11] I. H. Witten and E. Frank, Data Mining: Practical machine learning tools and techniques, 2nd ed. Morgan Kaufmann, 2005.

GNITED MINDS
Journals