Adaptive Progress Indicator for Long Running SQL Queries

MARIO MILICEVIC, KRUNOSLAV ZUBRINIC, IVONA ZAKARIJA
Department of Electrical Engineering and Information Technology
University of Dubrovnik
Cira Carica 4, 20000 Dubrovnik
CROATIA

Abstract: - Percent-done progress indicators are a technique for graphically showing how much of a long task has been completed. In the database environment such information is especially important during the long-running query execution. The proposed method constructs (during the learning phase) adaptive progress indicator model analyzing influence of averaged system state (represented with attributes describing CPU, memory and disk subsystem activity) on the query response time. Experimental evaluation shows that adaptive progress indicators can enhance users' experience and productivity.

Key-Words: - progress indicator, query response time, long-running query

1 Introduction

In recent years Data Warehousing (DW) and Business Intelligence (BI) technologies have grown as new hardware and software solutions lowered cost and simplified implementation. DW applications have focused on strategic decision support, leading to complex and often long-running queries - but in the same time the real-time enterprise initiatives imply interactive analysis that requires fast query response times. Many different tools and techniques are discussed and implemented to improve DW performances, but the importance of the user-system interaction (interface) is often neglected - especially details like progress indicators (estimators).

Even two decades ago, Myers [9] analyzed the importance of progress indicators on the user experience in graphical user interfaces. He concluded that users have a strong preference for the progress indicators during long tasks, because they enhance the attractiveness and effectiveness of programs that incorporate them. Progress indicators give the users enough information at a quick glance, to estimate how much of the task (not necessarily in database context) has been completed, and when the task will be finished.

Unfortunately, today's database systems provide only basic feedback to users about the query execution progress, especially regarding impact of the database server throughput on query response time.

2 Related Work

The basic considerations about response times has been discussed about thirty years ago, when Miller [7] described three important threshold levels of human attention.

Same limits were confirmed for the web based application in [10]:

- 0.1 second - the limit for having the user feel that the system is reacting instantaneously, meaning that no special feedback is necessary, except to display the result.
- 1.0 second - the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. Normally, no special feedback is necessary during delays of more than 0.1 but less than 1.0 second, but the user does lose the feeling of operating directly on the data.
- 10 seconds - the limit for keeping the user's attention focused on the dialogue. For longer delays, users will want to perform other tasks while waiting for the computer to finish, so they should be given feedback indicating when the computer expects to be done. Feedback during the delay is especially important if the response time is likely to be highly variable, since users will then not know what to expect.

Progress indicators in the database query processing context were discussed in the several recent works. Chaudhuri et al. [1] [2] introduced the concept of decomposing a query plan into a number of segments delimited by blocking operators. Query progress is estimated with the number of getnext() calls made by operators - but only in the context of an isolated system (where there is no other activity besides the execution of the query).

Similar approach was presented by Luo et al. [3] [4]. In order to support the progress indicators, authors divided a query plan into one or more segments and focused on the individual segments rather than the entire query plan, but again for the isolated single query. In the next work [5] they extended model with the multi-query

progress indicator, which explicitly considers concurrently running queries.

Mishra et al. [5] also presented framework for progress estimation of the operators and query segments using the getnext() model of query progress, with the similar limitations as mentioned before.

Majority of existing approaches relies on estimating intermediate cardinalities of operators in the query plan, but also requires communication with database engine during the query execution - which can be demanding or even unsupported.

3 Adaptive Progress Indicator

In general, percent-done progress indicators are a technique for graphically showing how much of a long task has been completed [9].

3.1 General Properties of Progress Indicators

Desirable properties of progress indicators were mentioned in [1]:

- Accuracy: the estimated percentage of work completed by the query at any point during its execution should be close to the actual percentage of work completed by the query at that point;
- Fine granularity: it follows from the above accuracy requirement that the estimator should be able to provide estimates at sufficiently fine granularity over the duration of the query's execution;
- Low overhead: an essential requirement for a progress estimator to be practical is that it should impose low overhead on the actual execution of the query.
- Leveraging feedback from execution: as query execution progresses, more information based on (intermediate) results of execution becomes available. Ideally, an estimator should be able to take full advantage of such information;
- Monotonicity: since the actual execution of the query progresses monotonically, ideally, the estimated progress should be also be monotonically increasing from the start of query execution to its finish.

While the first three properties are unquestionable, insisting on the last two features can produce compelling problems during the implementation phase. Leveraging feedback from the execution based on the intermediate results relies on (often problematic and/or undocumented) communication with RDMS during the query execution.

Monotonicity is logical requirement in the context of the isolated single query system, but the real-life examples show that database server workloads and resource utilization change considerably over time - resulting in the varying response times for identical queries. Furthermore, server's throughput can be dramatically changed (increased or decreased) during the long-running query execution - so corresponding response time predictions (as a function of the instantaneous server workload) also vary widely (but not always monotonically).

3.2 Model and Implementation

Influence of the system load and the system throughput on the response time, as well as a possibility of the accurate response time prediction - as a function of actual system state (CPU, memory and disk subsystem activity) during 1s interval, immediately before the observed query activation - has been already analyzed in [6].

In this work we consider another approach for constructing model representing SQL query behavior under dynamic server workload. During the learning phase, system state is again monitored within 1s intervals, but during the complete query execution. In the next step collected averaged system states (represented with the averaged attributes) and measured response times are taken as an input into different data mining algorithms [13]. Using linear regression we can build usable models, but better results can be expected with more elaborate methods (e.g. M5P [11] [12]). In order to capture the true system's behavior under various conditions, at least three structurally different queries must be analyzed - leading to at least three parallel models.

Built models are used during the production phase for the estimation (in regular intervals - default value is 1s) of the remaining query execution time - as a function of averaged system state (till observed moment). Response time prediction with three similar, but different models has a smoothing effect and the relative prediction error decreases considerably. Unknown queries from the production phase can be associated with referent queries from the learning phase through the estimated cost (supplied by Cost Based Optimizer (CBO)).

System state monitoring and models implementation during production phase fulfill low-overhead requirement.

3.3 Experimental Evaluation

Different long-running queries were observed in the environment of the real database server (up to 200 concurrent users).

Figure 1 shows distribution of the response times for one of analyzed queries. Response times vary between 92s and 472s, with typical response time of 110s (measured in single-query environment, with minimized influence of database cache memory). It is obvious that

progress indicator models built with the single-query prerequisite cannot be applied effectively.

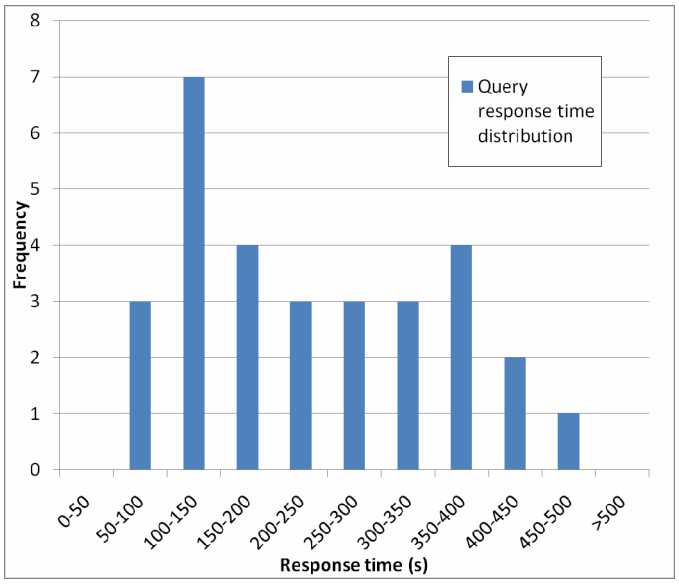

Fig. 1. Query response time distribution

Figure 2 shows implementation results of the proposed adaptive model. Measured (actual) response time is 226s. Ideal response time (110s) as a result of the single-query model is also specified. It can be noticed that predicted response times vary with server workload fluctuations. Initial predictions (during the first 30s of query execution) are underestimated as a consequence of the higher server throughput. Prediction response time variations can be smoothed using moving averages.

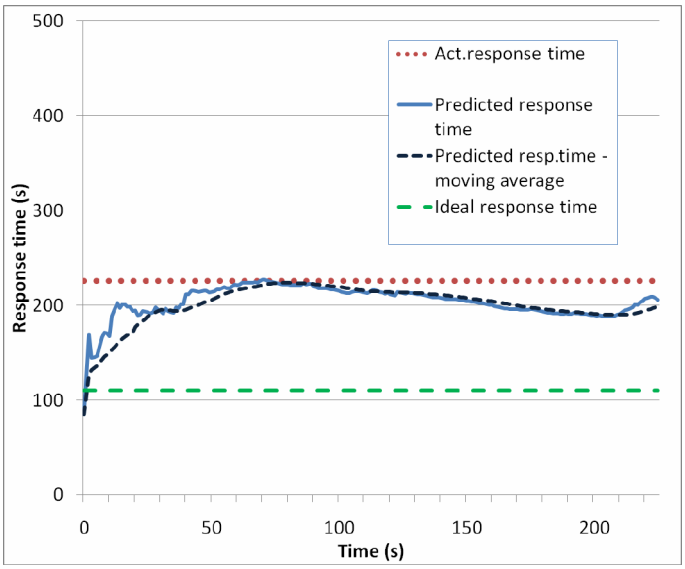

Fig. 2. Predicted response times during query execution

Behavior of the corresponding progress indicator is presented in Figure 3. - where it is obvious that progress indicator built on the (ideal) single-query model is unacceptable. Predicted progress - based on adaptive model - acceptably estimate real query progress. Query will be marked as 100% finished after 193s (what corresponds to 83% of real query progress). This result is quite acceptable taking into account that the single-query model proclaims query completed after only 49% of real query progress.

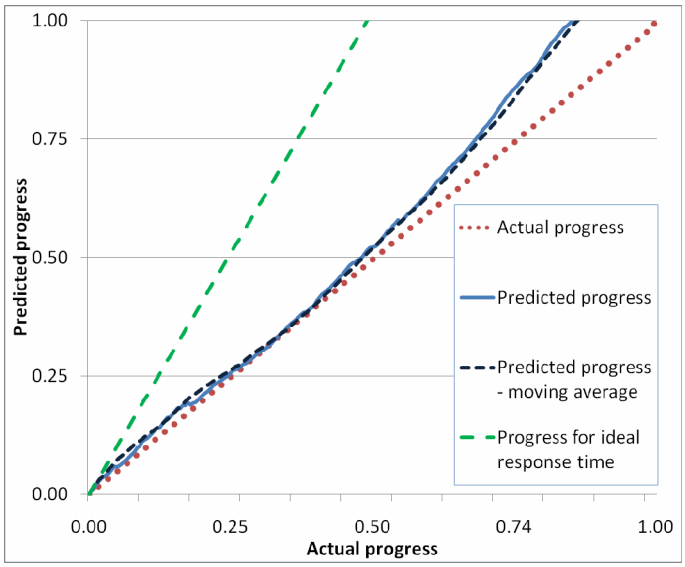

Fig. 3. Actual and predicted query progress

Figures 4 and 5 show identical query execution under different conditions (increased server workload). Measured (actual) response time is 362s. In this case, adaptive model lead to somewhat increased predicted response times, resulting in pessimistic predicted query progress: when query processing is completed, adaptive progress indicator shows 93% completion. Again, this result is acceptable taking into account that response time is 230% longer than under ideal conditions.

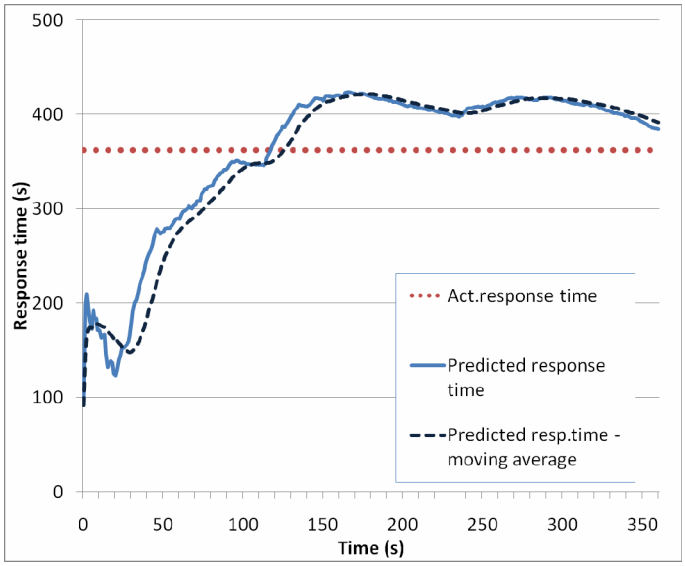

Fig. 4. Predicted response times during query execution

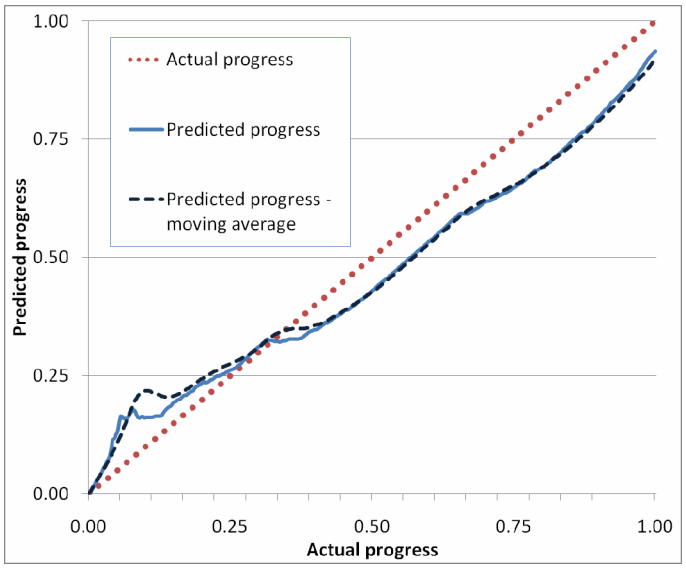

Fig. 5. Actual and predicted query progress

In the first example even monotonicity requirement is generally fulfilled, unlike the second example where pronounced increase of server workload (in several moments) leads to decrement of predicted query completion (e.g. from 22% to 20%).

4 Conclusion

Proposed method for adaptive progress indicator is based on the tracking of the system state changes (represented with adequate attributes). Models built in the learning phase predict query execution time (total and remaining). Existence of the learning phase can perhaps present a problem in some environments, but - on the other hand - the fact that there is no need for detailed knowledge about server configuration or query structure can be an important achievement.

References:

[1] S.Chaudhuri, V. Narassaya, R. Ramamurthy, Estimating Progress of Execution for SQL Queries. *SIGMOD*, 2004.

[2] S.Chaudhuri, R. Kaushik, R. Ramamurthy, When Can We Trust Progress Estimators for SQL Queries. *SIGMOD*, 2005.

[3] G.Luo, J.Naughton, C.Ellman, M.Watzke, Increasing the Accuracy and Coverage of SQL Progress Indicators, *ICDE*, 2004.

[4] G. Luo, J. Naughton, C. Ellman, M. Watzke, Toward a Progress Indicator for Database Queries, *SIGMOD*, 2004.

[5] G.Luo, J.Naughton, P.Yu, Multi-query SQL Progress Indicators, *EDBT*, 2006.

[6] M.Milicevic, M.Baranovic, V.Batos, *QoS control based on query response time prediction*, WSEAS Transactions on Computers 8, Vol. 4, 2005, pp. 882-889.

[7] R.B.Miller, Response time in man-computer conversational transactions, *Proceedings of AFIPS Fall Joint Computer Conference,* Vol. 33, 1968, pp. 267-277.

[8] C.Mishra, N.Koudas, A Lightweight Online Framework For Query Progress Indicators, *23rd International Conference on Data Engineering*, 2007, pp. 1292-1296.

[9] B.A.Myers, The importance of percent-done progress indicators for computer-human interfaces, *Proceedings of ACM CHI'85 Conf.*, 1985, pp.11-17.

[10] J.Nielsen, *Usability Engineering*, Morgan Kaufmann, San Francisco, 1994.

[11] J.R.Quinlan, Learning with continuous classes, Proceedings AI'92, ed., Sterling Adams, Singapore 1992., pp. 343-348.

[12] Y.Wang, I.H.Witten, Induction of model trees for predicting continuous classes, *Proceedings of the Poster Papers of the European Conference on Machine Learning*, Prague 1997., pp.128-137.

[13] I.H.Witten, E.Frank, *Data Mining: Practical machine learning tools with Java implementations*, Morgan Kaufmann, 2000.