INTERNATIONAL JOURNAL OF COMPUTERS
Issue 4, Volume 2, 2008

# Dynamic Approach to the Construction of Progress Indicator for a Long Running SQL Queries

Mario Milicevic, Krunoslav Zubrinic, Ivona Zakarija

***Abstract*—**Widely used Data Warehousing (DW) and Business Intelligence (BI) technologies are mostly based on complex and long-running queries. It is very important for users to have information about query execution progress. Percent-done progress indicators are a technique that graphically shows query execution time (total and remaining) or degree of completion. In this paper we propose a method that constructs model of a percent-done progress indicators based on adaptive approach. During the learning phase, the method analyzes the influence of averaged system state on the SQL query response time using data mining algorithms. The results of this phase are models representing query behavior under dynamic server workload. Built models are used during the production phase for the estimation of query execution time. Experimental evaluation confirms the effectiveness of created models.

***Keywords*—**long-running query, progress indicator, query response time

## I. Introduction

IN recent years Data Warehousing (DW) and Business Intelligence (BI) technologies [3] have grown as new hardware and software solutions lowered cost and simplified implementation [4]. 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 [12] 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.

Nielsen [13] concluded that progress indicators have three main advantages: they reassure the user that the system has not crashed but is working on his or her problem; they indicate approximately how long the user can be expected to wait, thus allowing the user to do other activities during long waits; and they finally provide something for the user to look at, thus making the wait less painful. This latter advantage should not be underestimated and is one reason for recommending a graphic progress bar instead of just stating the expected remaining time in numbers.

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.

## II. Related Work

Computer system response time is generally defined as the number of seconds it takes from the moment users initiate an activity until the computer begins to present results on the display or printer [16]. There are still no industry standards for acceptable application response time.

The basic considerations about response times in the context of Human-Computer Interaction (HCI) has been discussed about thirty years ago, when Miller [10] described three important threshold levels of human attention.

Same limits were confirmed for the web based application in [13]:

- 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

M. Milicevic is with the Department of Electrical Engineering and Information Technology, University of Dubrovnik, 20000 Dubrovnik, Croatia (e-mail: mario.milicevic@unidu.hr).

K. Zubrinic is with the Department of Electrical Engineering and Information Technology, University of Dubrovnik, 20000 Dubrovnik, Croatia (e-mail: krunoslav.zubrinic@unidu.hr).

I. Zakarija is with the Department of Electrical Engineering and Information Technology, University of Dubrovnik, 20000 Dubrovnik, Croatia (e-mail: ivona.zakarija@unidu.hr).

computer expects to be done. System indicators should give them information that the system is working on his or her problem. They indicate approximately how long the user can be expected to wait, allowing him or her to do other activities during longer delays. 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). Authors also discussed what the important parameters are, and under what situations robust progress estimation can be expected.

Similar approach was presented by Luo et al. [5], [6]. 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 [7] they extended model with the multi-query progress indicator, which explicitly considers concurrently running queries and even queries predicted to arrive in the future when producing its estimates. Authors also demonstrated how to apply the resulting multi-query progress indicators to several workload management problems.

Mishra et al. [11] 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. Proposed method can estimate progressively the output size of various relational operators and pipelines, minimal overhead on query execution. These include binary and multiway joins as well as typical grouping operations and combinations thereof.

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.

## III. Adaptive Progress Indicator

In general, percent-done progress indicators are a technique for graphically showing how much of a long task has been completed [12].

### A. 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 can become 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 RDBMS during the query execution.

Monotonicity is logical requirement in the context of the isolated single query environment, 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).

### B. Model and Implementation

Considerations and options related to quantitative information for generic progress indication displays has been discussed in [8]:

- Providing quantitative progress information, or only a busy indicator, such as an hour glass;
- Presenting progress in terms of percentages, or as a count of completed items. For example: 90% complete or 5 out of 6 steps complete;
- Measuring time vs. steps. For example: 30 seconds vs. 2 steps;
- Considering the amount remaining or time elapsed. For example: 10% done or 10% remaining;
- Displaying progress of the current step, or overall progress. For example: Step 1 is 90% complete or 90% complete overall;
- Displaying rate information. For example: 3K/second;
- Accuracy of estimates. For example: displaying 5 seconds when it is really 10 minutes;
- Precision of estimates. For example: 3.4% remaining vs. about 10% remaining.

Authors also discussed implementation requirements for process progress in a relatively smooth, linear fashion. Accurate progress estimation is often difficult to achieve; in

these cases initial estimate and ongoing re-estimates are required. Therefore, it needs to be considered how accurate an initial estimate can be and whether periodic re-estimates are required. This needs to be balanced with the performance overhead of such estimates.

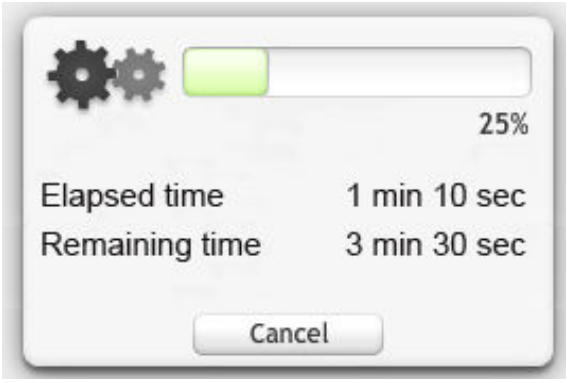


Fig. 1 Simple progress indicator

Simple progress indicator (Fig. 1) - in the query processing environment - must inform users at least about elapsed and remaining query processing duration.

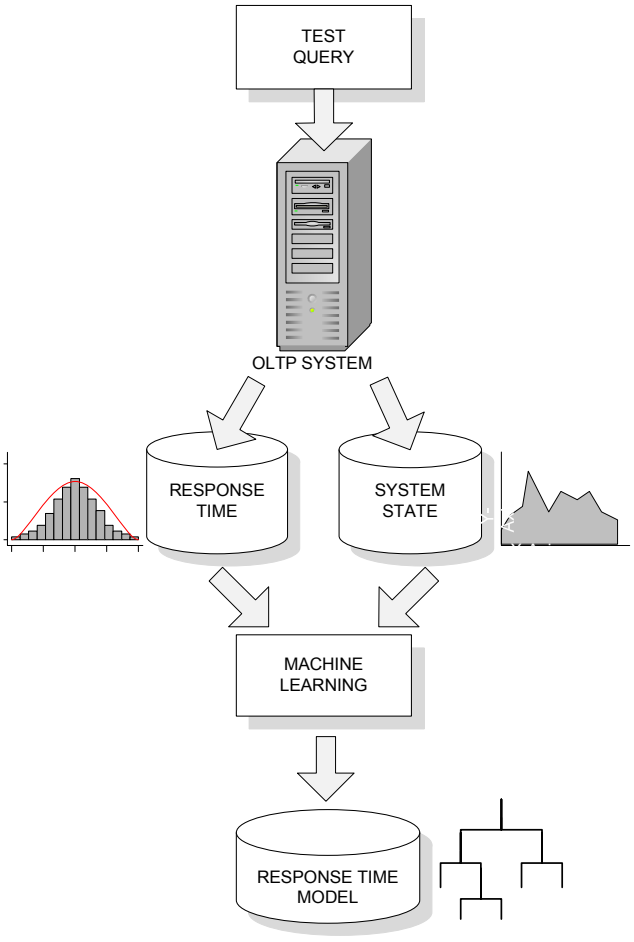


Fig. 2 Learning phase

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 [9].

In this work we consider another approach for constructing model representing SQL query behavior under dynamic server workload. During the learning phase (Fig. 2) 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 [14], [18]. Already using linear regression (1) we can build usable models:

$$\begin{aligned} query\_duration &= 8.0555 * b - 0.0002 * avm \\ &+ 0.0497 * free - 0.2081 * po - 0.1446 * fr \\ &- 0.0047 * sr + 0.0832 * in - 0.0058 * sy \\ &- 0.0295 * cs + 6.7706 * us + 147.4438 * sys \\ &+ 4.1851 * wait + 5.5608 * hdD \\ &+ 1.1118 \end{aligned} \quad (1)$$

where attribute selection method - for presented referent query - choose subsequent attributes (among more than 30 monitored attributes):

b - average number of kernel threads placed in the VMM wait queue;
avm - active virtual pages;
free - size of the virtual pages free list;
po - pages paged out to paging space;
fr - pages freed (page replacement);
sr - pages scanned by page-replacement algorithm;
in - device interrupts;
sy - system calls;
cs - kernel thread context switches;
us - user time;
sys - system time;
wait - processor idle time during which the system had I/O request(s);
hdD - activity of HD(s) with data tablespaces.

Better results can be expected with more elaborate methods, e.g. M5P (Model Trees) [15] [17].

M5P is a regression tree algorithm designed for continuous classes. First, an ordinary classification tree is constructed, with the standard deviation reduction used as node impurity function. Then the tree is pruned (controlling a tradeoff between prediction and tree size), with a stepwise linear regression model fitted to each node at every stage (rather than to simply predict the mean).

A final stage is to use a smoothing process [17] to compensate for the sharp discontinuities that will inevitably occur between adjacent linear models at the leaves of the pruned tree, particularly for some models constructed from a small number of training instances. The smoothing procedure described by Quinlan [15] first uses the leaf model to compute the predicted value, and then filters that value along the path

back to the root, smoothing it at each node by combining it with the value predicted by the linear model for that node.

The tree could be simplified adding a restriction about the minimum number of instances (default 4) covered by each leaf node (larger value will generate a simpler tree but less accurate).

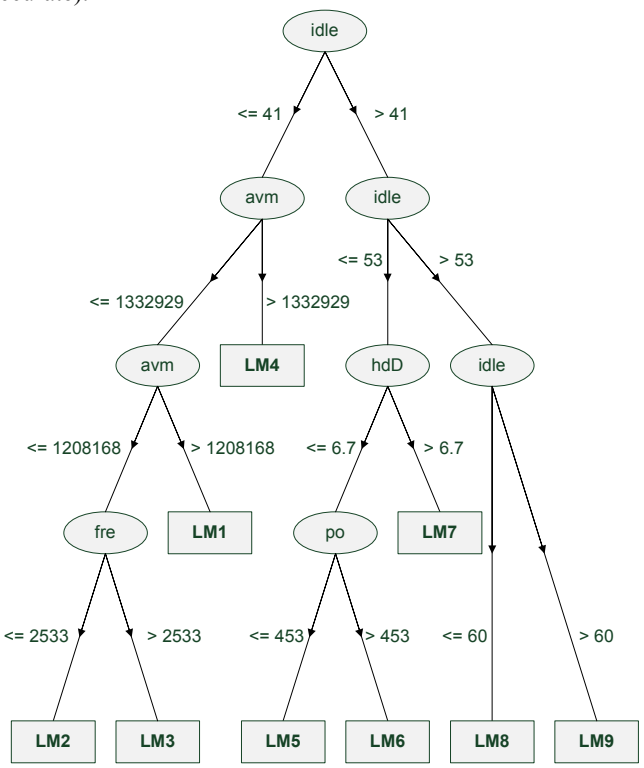


Fig. 3 M5P decision tree

In the M5P decision tree example shown in Fig. 3, the M5P algorithm created 9 leaves (rules, linear models):

LM1
duration = - 0.0001 * avm
- 4.1649 * idle
+ 587.4366

LM2
duration = - 0.0001 * avm
- 4.1649 * idle
+ 589.3916

LM3
duration = - 0.0001 * avm
- 4.1649 * idle
+ 589.2693

LM4
duration = - 0.0001 * avm
- 4.1649 * idle
+ 575.3043

(2)

LM5
duration = - 0.0001 * avm
- 0.0599 * po
- 5.1319 * idle
- 1.4934 * hdD
+ 564.3411

LM6
duration = - 0.0001 * avm
- 0.0599 * po
- 5.1319 * idle
- 1.4934 * hdD
+ 563.8036

LM7
duration = - 0.0001 * avm
- 0.0574 * po
- 5.1319 * idle
- 1.9082 * hdD
+ 563.7388

LM8
duration = - 0.0001 * avm
- 0.0552 * po
- 5.9547 * idle
+ 585.8962

LM9
duration = - 0.0001 * avm
- 0.0552 * po
- 5.9054 * idle
+ 582.1818

As it can be noticed, model tree usually uses only a small subset of the available attributes. Besides already mentioned thirteen attributes (1), two new attributes appear in linear models LM1-LM9:

idle - processor idle time;
po - pages paged out to paging space.

In order to capture the actual system's behavior under various conditions, at least three structurally different queries must be analyzed - leading to at least three parallel models. Each model predicts response time, while overall prediction is result of averaging.

Chosen referent queries must represent server load faithfully. This requirement can be fulfilled by analyzing DB logs and query frequency, using prior knowledge about application structure, or by detecting and analyzing users' behavior patterns.

The learning phase must be conducted during several days or weeks. All test queries have been executed more than 50 times each - during all characteristic periods related to the rhythm of typical users' activities. Also all attributes presenting system state were recorded during test queries executions.