Tracking Query Progress

Seminar: Databases (Winter Term 2022/2023)

YECINE MEGDICHE

This seminar paper presents a proof of concept for tracking query progress in the Umbra database [5]. Query progress is a helpful metric for users, as it helps identify errors and informs decisions to interrupt query execution. It also may provide a hint to the database management system to adjust scheduling strategy for long-running queries. Instead of estimating the time left for a query to finish, the developed estimators report the percentage of work done at the estimation time. Three different estimators for tracking query progress are implemented and evaluated based on their accuracy gainst the TPC-DS [2] dataset. The results show that the weighted finished pipelines with job progress estimator is the most accurate.

1 INTRODUCTION

Analytical workloads on database systems can have long running queries. Except for some commercial database systems like Microsoft SQL Server [1], most systems do not provide feedback to users on the progress made by the query. As a result, users cannot accurately predict the duration still needed for their workload to finish.

The query progress is a practical metric for users. For example, if a query exhibits unexpectedly slow progress, then users can interrupt the execution and check the statement for potential errors. It is also a useful metric for the database management system, since the scheduler can predict long-running queries as the queries that make little progress after a certain amount of time, for instance, and adjust the scheduling strategy accordingly.

The most effective metric for users is the duration left for a query to finish. However, this metric is fundamentally hard to estimate. Progress estimation inherits the difficulties of cardinality estimation: the estimator should guess the total amount of work to estimate how much work is left. Additionally, the execution time also depends on external factors like the number of queries concurrently running in the system or network latency. Thus, instead of estimating the time left, the database system can instead report the amount of progress a query has made.

We implement a proof of concept for tracking query progress in Umbra [5]. The implementation includes three estimators for the percent of work achieved by the query. Finally, we evaluate the accuracy of the estimators against the TPC-DS dataset [2].

2 RELATED WORK

One of the first published work on estimating query progress [1] is implemented on Microsoft SQL Server. Their estimation is based on “driver nodes”: nodes in the execution tree that provide the data for the rest of a pipeline. These nodes are typically either table scans or pipeline breakers like sort nodes. As a result, it easy to estimate the work done by each them since cardinalities are known accurately. The claim is that the progress of driver nodes is an accurate estimate for the progress of the whole pipeline.

3 BACKGROUND

3.1 Relation Algebra and Execution

A SQL Query is transformed into relational algebra to be executed by the system. Relational algebra represents an operational view of the query, in contrast to the declarative aspect of SQL. A relational algebra expression is composed of operators, and a set of operators that do not require intermediate materialization of results is called a pipeline.

Umbra uses morsel-driven parallelism for executing a pipeline [3]. A job abstraction manages the morsels needed for each pipeline, allowing worker threads to pick and execute a morsel [4]. It also provides an interface to estimate the progress of the pipeline based on the scheduled morsels.

3.2 Estimator properties

An estimator should have the following properties [1]:

- Accuracy: the estimated progress of a query should be comparable to its actual progress.
- Fine granularity: the estimated progress should have a sufficiently fine granularity.
- Low overhead: the progress estimation should not be computationally expensive.
- Leveraging feedback from execution: the query estimator should use as much information available from the execution as possible.
- Monotonicity: the progress estimations reported for a query should be non-decreasing.

4 ESTIMATORS

We implement the following progress estimators:

(1) **Finished Pipelines Estimator** (abbreviated: FP): returns the number of finished pipelines divided by the total number of pipelines.
(2) **Weighted Finished Pipelines Estimator** (abbreviated: WFP): returns the sum of the estimated cardinality input of each finished pipeline during query planning divided by the sum of the estimated cardinality of all pipelines. Cardinality estimates are computed during query planning.
(3) **Weighted Finished Pipelines with Job Progress Estimator** (abbreviated: WFPJ): returns the estimation from the weighted finished pipelines estimator, with the addition of the progress of the current pipeline. The pipeline progress corresponds to the job progress weighted by the estimated pipeline input cardinality.

Per design, all estimators are monotone. The WFPJ estimator has arbitrary granularity, whereas the other two are limited. The FP estimator can only report values in increments of $1/\#pipelines$ for a given query. The WFP estimator also has fixed values that it can return: the increments are determined by the cardinality estimates from the query optimization phase.

Moreover, the first two estimators do not leverage enough information from the execution: they only track when a pipeline is finished. However, the WFPJ estimator updates its estimates whenever a new morsel is scheduled.

5 IMPLEMENTATION

The execution plan, an object representing a query during runtime, is extended to include properties tracking query progress estimates. These are updated whenever needed: when a pipeline is started or finished, or when a new morsel is scheduled. This ensures that the estimators can be used with minimal overhead, since both update and read operations are fast.

The running queries are tracked in the database. The queries are extended with a unique identifier. Moreover, the session has an extra setting `query_name` to identify queries issued by that session.

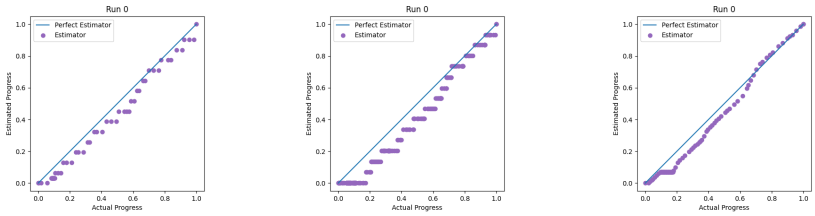
Query progress is reported on demand: a system table `um_progress` can be queried with a SQL statement. For each query in the system, the table contains a row with an id, its estimated progress, its current runtime, the session's query name, and whether it finished execution. Once the query is finished running and is read from `um_progress` the database stops tracking that query.

The estimators can be chosen with the setting `progressestimator`. This setting must be set per database instance and not session specific.

6 EVALUATION

6.1 Experiment

- **Goal:** The goal of the experiment is to evaluate the accuracy and compare the estimators.



(a) FP (b) WFP (c) WFPJ

Fig. 1. Query 9 results with the different estimators

- **Setup:** The experiment is ran on a machine with an Intel® Xeon® CPU E5-2680 v4 @ 2.40GHz processor and with 256GiB of RAM.
- **Dataset:** The expirment is ran on the TPC-DS dataset [2] with scale factor 100.
- **Experiment:** Once the umbra server is started, the benchmark connects to the server with two threads: one thread monitors the query progress by constantly querying `um_progress`, and the other runs the queries used for evaluation. The thread sets the session name to the query number and runs it three times.

6.2 Metric

As described in [1], each estimator is compared to a perfect estimator. A perfect estimator has perfect knowledge of the future, and thus when queried at time t_i would report a progress of $\frac{t_i - t_0}{t_n - t_0}$ for a query starting at time t_0 and finishing at t_n. If an estimator reports f_i as the progress at time t_i, then its error e_i is:

$$e_i = \left| \frac{t_i - t_0}{t_n - t_0} - f_i \right|$$

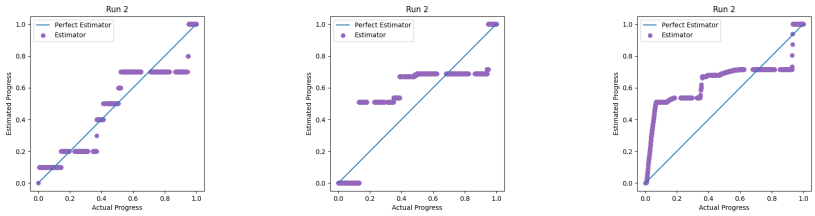
6.3 Results

Estimator	Mean Squared Error	Average Error	Max Error	Number of queries with Least Average Error
FP	0.06	19.14%	90.4%	24
WFP	0.16	30.58%	99.54%	7
WFPJ	0.03	12.36%	78.35%	71

Table 1. Estimator Performance

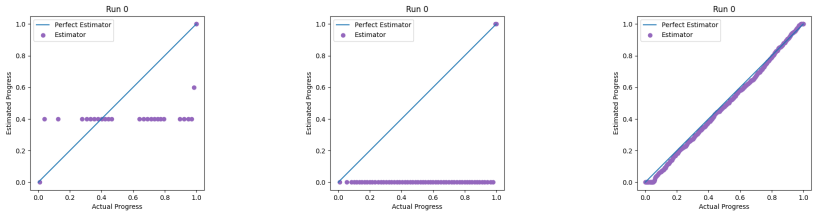
Table 1 summarizes the results for the three estimators. On average, the WFPJ progress estimator had the best overall metrics: it performed best on 71 out of the 102 queries used for the benchmark, had the least overage error 12.36%, and the lowest maximum error of 78.35%.

For queries with many pipelines that perform a similar amount of work, the three estimators have similar results. As an example, query 9 has 15 pipelines that all start scan the table `store_sales` and perform different aggregations. The results can be seen in figure 1.



(a) FP (b) WFP (c) WFPJ

Fig. 2. Query 51 results with the different estimators



(a) FP (b) WFP (c) WFPJ

Fig. 3. Query 52 results with the different estimators

Despite the simplicity and the limited granularity of the FP estimator it can still deliver better estimates than the other two estimators. This is in fact the case in 51 of the queries as seen in figure 2. Note that in this example WFPJ overestimates the progress in the first pipelines.

The WFP estimator, however, has the worst metrics. It has a maximum error of close to 100%: for some queries, it estimates a progress of close to 0% even when the query is almost finished. This happens for queries where a single pipeline dominates: the pipeline's estimated cardinality and runtime overshadow the rest. As a result, as long as that pipeline is running (which is almost the query's entire lifetime), the estimator reportes no progress. In contrast to that behavior, the WFPJ provides accurate estimates for these queries: Umbra can accurately estimate the progress of such long-running pipelines, and their progress is usually enough to estimate the query's overall progress. An example for such a query is query 52, which can be seen in figure 3. Query 52 consists of 5 pipelines where the third pipeline dominates the runtime. The WFP estimator reports a maximum error of 97.9%, compared to 5% for WFPJ. Note that the FP estimator outperforms the WFP estimator since it gives equal weight to all pipelines, which results in a progress of 40% as long as the third pipeline is running.

Although the WFPJ estimator continuously refines its estimates, it does not always capture the full context. A notable aspect is that it does not consider whether a table is cached by the database, which can have a substantial impact on the runtime of the query. This can be observed in query 28: the first run took 1.74 seconds and had an average error of 29.09%, whereas the second one took only 0.51 seconds and had an average error 0.03%. The results can be seen in figure 4.

7 CONCULSION AND FUTURE WORK

In this paper, we outline the importance of query progress estimation and implement three estimators in the Umbra database: the finished pipelines estimator, the weighted finished pipelines estimator, and the weighted

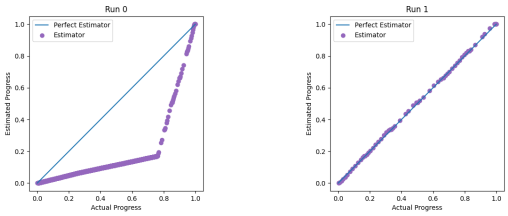


Fig. 4. Query 28 with WFPJ estimator

finished pipelines estimator with job progress. These estimators are benchmarked against the TPC-DS Dataset with a scale factor of 100 to evaluate their accuracy. Overall, the weighted finished pipelines estimator with job progress provided the best results.

The evaluation dataset could also be expanded to study the different aspects that influence the accuracy of the estimators. A notable shortcoming of the dataset and the scale factor used was that the queries did not have a long runtime: only 19 had an average runtime of more than one second, with a longest runtime of 11.4 seconds.

The progress estimation can be improved by implementing estimators that leverage more information from the runtime system. For example, one estimator could refine the cardinality estimates during runtime and not only rely on the estimates from the query planning phase. One could also consider other properties like caching to adjust the weights of the different pipelines.

The progress tracking system can be improved by adding access control to `um_progress`, so a user only sees queries that are added by them. Moreover, the logic that stops tracking queries and prevents them from appearing in `um_progress` does not consider filters that occur when constructing `um_progress`. In its current form, it allows a convenient interface to benchmark and evaluate the estimators is not ready for a production setting.

REFERENCES

[1] Surajit Chaudhuri, Vivek Narasayya, and Ravishankar Ramamurthy. 2004. Estimating progress of execution for SQL queries. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, New York, NY, USA, 803–814. https://doi.org/10.1145/1007568.1007659

[2] Transaction Processing Performance Council. 2015. TPC-DS Specification. http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.10.0.pdf.

[3] Viktor Leis, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2014. Morsel-driven parallelism: a NUMA-aware query evaluation framework for the many-core age. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 743–754.

[4] Thomas Neumann. 2021. Evolution of a Compiling Query Engine. *Proc. VLDB Endow.* 14, 12 (jul 2021), 3207–3210. https://doi.org/10.14778/3476311.3476410

[5] Thomas Neumann and Michael J. Freitag. 2020. Umbra: A Disk-Based System with In-Memory Performance. In *10th Conference on Innovative Data Systems Research, CIDR 2020, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2020/papers/p29-neumann-cidr20.pdf