

Compressing Very Large Database Workloads for Continuous Online Index Selection ^{*}

Piotr Kolaczowski

Warsaw University of Technology, Institute of Computer Science
P.Kolaczowski@ii.pw.edu.pl

Abstract. The paper presents a novel method for compressing large database workloads for purpose of autonomic, continuous index selection. The compressed workload contains a small subset of representative queries from the original workload. A single pass clustering algorithm with a simple and elegant selectivity based query distance metric guarantees low memory and time complexity. Experiments on two real-world database workloads show the method achieves high compression ratio without decreasing the quality of the index selection problem solutions.

Key words: database workload compression, automatic index selection

1 Introduction

Efficient solution of the *index selection problem* (ISP) usually requires the calculation of a set of queries approximating the database workload. There are many algorithms for solving the ISP [1–8]. The more queries are given at their input, the more resources and time they need to find good results. As it is inconvenient to create the input data by hand, database administrators usually use query logs to get the approximation of the future workload. Unfortunately the query logs can be very large. Millions of queries logged by a transactional system during a day are not uncommon. Without compression, solving the ISP for all the logged queries may take unreasonably large amounts of time. For the purpose of practical usability of the automated physical database tuning tools, it is crucial to reduce the number of the analyzed queries. The simplest solution would be to pick only a small random sample of the workload. However, this method might skip some important queries by accident and in effect significantly degrade the results given by the database tuning tool. A good method for workload compression should not have such side effects. It should achieve high compression ratio while keeping the quality of the results at the acceptable level.

For the purpose of the experimental analysis we define the *quality ratio* as the ratio between the original workload costs after automatic selection of indexes with and without using the workload compression. The *quality loss* is the difference between these two costs. The ideal workload compression algorithm would

^{*} The work has been granted by Polish Ministry of Education (grant No 3T11C 002 29)

have the quality ratio of 1 and quality loss of 0. The most known algorithms cause some quality loss and their quality ratio is less than 1.

The *online ISP* is a more difficult variant of ISP, where indexes can be created or dropped at any time and the workload is not known in advance, but should be treated as a stream of continuously incoming queries. Solving this problem is crucial for building self-managing, autonomic database systems [9, 10]. Efficient solving of the online ISP requires different workload compression algorithms that can compress the workload *incrementally*, “on the fly”.

We present an algorithm that removes most of the queries from the workload and leaves only those essential to solve the online ISP accurately and efficiently. To achieve this, the algorithm employs single-pass clustering with a selectivity based metric. Because each query is analyzed only once, the method can be used to improve the performance of continuous online database tuning systems [11–13]. To the best of our knowledge, our method is the first one that requires only one pass to efficiently compress large SQL workloads while keeping high quality of results of the ISP.

2 Previous work

The problem of SQL workload compression was first stated by S. Chauduri et al. [14]. They proposed three different algorithms for compressing SQL workloads: a variant of random sampling (WC-PARTSAMP), a variant of the well known K-Medoids algorithm (WC-KMED) and a greedy algorithm removing queries from the workload as long as the results quality constraint is satisfied (WC-ALLPAIRS). Of these algorithms, only the WC-PARTSAMP requires no more than one pass over the input data, but has several shortcomings typical to random sampling methods. Both the compression ratio and the ISP solution quality can vary significantly for different runs of the program. There is also no way of reasonably setting the sampling rate without knowing the characteristics of the workload in advance. For large workloads with lots of similar queries, the sampling rate should be small, but for workloads with many different queries it should be large enough not to miss the important queries. The WC-KMED and WC-ALLPAIRS algorithms don’t have these shortcomings and achieve higher compression ratio than the WC-PARTSAMP, but their time complexity is also much higher. Moreover, they need to be given the whole workload in advance. The WC-KMED algorithm invokes the standard k-Medoids (KMED) algorithm several times to guess the best number of clusters. Each invocation of KMED requires scanning the whole dataset many times, until the medoids stop changing. The WC-ALLPAIRS algorithm has even higher complexity because it must calculate the distance between every pair of queries in the workload. This makes these algorithms unsuitable for solving the continuous ISP.

The paper [14] also proposes an asymmetric query distance function used to estimate the quality loss of the ISP results caused by replacing a given query in the workload with another query. Although they put lot of effort into assuring their metric does not underestimate the quality loss, there are several situations

possible, where it actually does. This is caused by the simplified assumption that the index selection tool would select indexes for the columns used in the predicates with the lowest selectivity. As it is true in many cases, it is not true in general. First, using a clustered index for a less selective predicate can be cheaper than using a non-clustered index for a predicate with a higher selectivity, because accessing the tuples pointed by the clustered index usually requires less I/O operations than accessing the same number of tuples pointed by the non-clustered index [15]. The decision what index should be clustered is usually based on many queries in the workload and is not known at the time of the workload compression. Second, automatic index selection tools evaluate not only the benefits of indexes but also their maintenance costs. Thus, the index on the columns of the most selective predicate may have higher maintenance costs and be discarded. This problem applies also to covering indexes, where having a smaller index beneficial to only one query may be a better solution than having a larger index covering two queries but requiring much more maintenance. We propose a different metric that addresses these shortcomings and additionally has a symmetry property and satisfies the triangle inequality.

3 Workload Compression Algorithm

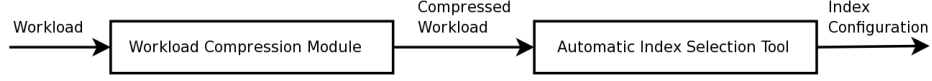
As shown in [14], the workload compression problem can be treated as a special case of a clustering problem, where each data point is a query in the workload and the distance between two points is a function of predicted ISP results quality loss if replacing one query with the other one. The queries in the workload are clustered and afterwards one query in each cluster is retained. The retained queries form the compressed workload. The number of queries in the compressed workload equals the number of the clusters, so the clustering algorithm should create as few clusters as possible to achieve good compression ratio. On the other hand, the queries in each cluster should be similar to each other. The more similar are the queries in each cluster, the less degradation of the ISP results is expected.

The requirement for incremental compression makes classic clustering algorithms k-Medoids, k-Means and hierarchical algorithms AGNES, DIANA, BIRCH, Chameleon [16] unsuitable for the purpose of online ISP. The density based algorithms like DBSCAN [16] are neither applicable, because they don't limit the maximum distance between the queries in each cluster. This could result in a large degradation of the ISP results.

The architecture of our solution is shown in Fig. 1. The incoming workload W is a stream of SQL queries. Each query q has a weight $w(q)$ assigned to it. The workload compression module supplies a compressed workload W' to the input of the tool that selects indexes. The compressed workload contains a subset of queries from W , but they can be assigned different weights w' so that the total estimated cost of W' is near the estimated cost of W .

The workload compression module uses internally a simple yet efficient algorithm that groups incoming queries in clusters (Fig. 2). A *seed query* is the first

Fig. 1: Automatic index selection improved by workload compression



query added to the cluster. The first incoming query becomes the seed of the first cluster. For each subsequent incoming query from W , the nearest seed query s is found. If the distance between these two queries exceeds a user-defined limit, then the incoming query forms a new cluster and becomes its seed. Otherwise the query is added to the cluster having seed s . The compressed workload is formed from the cluster seeds. Each cluster seed is assigned a weight evaluated as the sum of the weights of all queries in this cluster. Actually, to improve performance, only the seeds with weights are stored in memory, and instead of adding non-seed queries to the clusters, only the seed weights are updated.

Input: workload W , constraint δ , distance function d , query weights w

Output: compressed workload W' , modified weights w'

1. $W' := \emptyset$
2. For each query $q \in W$ do:
3. Set $w'(q) := w(q)$
4. If $W' = \emptyset$, add q to W' and continue
5. Find the nearest seed query: $s := \arg \min_{q' \in W'} d(q', q)$
6. If $d(s, q) > \delta$, add q to W'
7. Else $w'(s) := w'(s) + w(q)$

Fig. 2: Workload compression algorithm

Note that we do not use the distance function to adjust the weights of the output queries, like it was done in [14]. We argue, this is not needed because of the following reasons:

- Without actually running the index selection tool on both compressed and uncompressed workloads, it is not possible to evaluate the quality loss caused by removing a given query from the workload.
- It is not known in advance, which indexes would be created for the removed query by the index selection tool, so the estimations of the result quality loss may have large errors.
- If the δ constraint is small enough, the queries in each cluster would be very similar to each other and the difference between the estimated costs of the original and compressed workload would be also small.

4 Query distance function

The intuition behind the distance function is that the distance between query q_1 and q_2 should be small, if replacing the query q_1 with q_2 in the workload causes small quality loss. For example the queries:

```
SELECT * FROM person WHERE id = 12345
SELECT * FROM person WHERE id = 54321
```

are very similar to each other, assuming they select at most one tuple. Regardless of the index configuration, they will have the same query plans. Leaving one of them out of the compressed workload would cause no quality loss, as they both deliver the same information on the usefulness of the index on the `id` column. However, if the selectivities of the predicates in these queries were different, e.g.:

```
SELECT * FROM person WHERE age = 30
SELECT * FROM person WHERE age > 30
```

The queries look similarly, but they contribute different information on the usefulness of the index on the `age` column. The first query would be probably accelerated the most by the hash index on the `age` column. However, this index would not be useful for the second query. Probably the best choice for the second query would be some kind of clustered B-tree index, which could also serve the first query, but the availability of such clustered index is strongly dependent on the other queries in the workload. It is not possible to guess the solution, so it is wise to leave both queries in the compressed workload and let the index selection tool decide. The problem becomes even more complex if queries with more predicates and tables are concerned. The number of useful index configurations for such queries grows exponentially with the number of predicates. Thus, removing any of two queries differing significantly with at least one predicate selectivity may bias the workload against using some good index configurations.

The query distance d between queries q_1 and q_2 is evaluated as follows. If the queries differ in *structure*, that is with anything except constant literals:

$$d(q_1, q_2) = +\infty,$$

else

$$d(q_1, q_2) = \max_{p_1 \in \text{Pred}(q_1), p_2 \in \text{Pred}(q_2), p_1 \sim p_2} \frac{\max\{\text{Sel}(p_1), \text{Sel}(p_2)\}}{\min\{\text{Sel}(p_1), \text{Sel}(p_2)\}} - 1,$$

where:

- $\text{Pred}(q)$ is the set of the selection predicates in the query q
- $\text{Sel}(p)$ is the selectiveness of the predicate p in range from 0 to 1.
- $p_1 \sim p_2$ denotes two corresponding predicates, that differ only with the constant literals.

This metric has several advantages. It is symmetric, satisfies the triangle inequality and is very easy to implement. For most query pairs it is even not needed to evaluate the selectivities of the predicates, because differences in the query structure (set of tables, join predicates, number of selection predicates, column set in the `ORDER BY` or `GROUP BY` clause, etc.) can be easily spotted by shallow text analysis. This is very important, because for large workloads containing lots of simple queries, the parsing and plan generation process can become a bottleneck. Another advantage is that the proposed metric can leverage the existing support for prepared statements in most modern database systems. The comparison of the structure of queries can be performed at the time of statement preparation, not their execution.

5 Experiments

For the experiments, we used two real-world server side transactional applications: a commercial multiplayer network game with 100,000 users further referred as MG and a mobile web application of one of Polish telecom operators (WA). The MG executed much more update statements than WA, which was mostly read-only (Tab. 1). Besides, MG was a mature application being for over 3 years in production and used 108 tables, while WA application was in its beta-stage and used only 33 tables. Both applications sent EJB-QL queries and employed an object-relational mapping tool to convert them into valid SQL statements. The workloads did not contain any subqueries, however some of the queries required joining up to 6 tables and contained up to 10 selection predicates.

The framework for the workload compression has been build as a standalone application employing an ANTLR generated SQL parser and a histogram based predicate selectivity estimator. The histograms were imported from the statistics gathered by the database optimizer. For some of the queries, we manually compared our predicate selectivity estimation with the estimations made by the database optimizer in the `EXPLAIN` mode and noticed none or negligible differences.

As the automatic index selection tool we used a prototype tool developed at our institute. The tool can select single and multicolumn B-tree indexes, both clustered or unclustered, and takes index maintenance costs into account. The selection tool uses the same selectivity estimates the workload compression application does.

The measurement of the compression ratio for both workloads shows that the compressed workload size grows very slowly with the size of the input workload (Fig. 3). The index selection tool could not finish the computations in the given 1 hour period for the uncompressed workload of the MG application containing 25000 queries, but managed to finish the task in less than 2 minutes for the compressed workload.

As stated in Introduction, the workload compression algorithm is useful if both the compression ratio and quality ratio are high. We compared our method with other algorithms that could compress workloads incrementally: the random

Table 1: Characteristic of the workloads used in the experiments

Statement type	Share [%]	
	MG	WA
Single-table SELECT	66.95	78.73
Multi-table SELECT	18.93	16.14
Aggregate SELECT	2.84	3.30
INSERT	0.92	1.83
UPDATE	12.71	0.98
DELETE	0.49	2.31

sampling method, and the random sampling method preceded with partitioning used in [14]. To be able to calculate the quality-loss in a reasonable time, we had to limit the number of queries in the test workloads to 2000. Our compression method resulted in both good compression-ratio and quality-ratio. The results for the MG application are shown in Fig. 4, and the results for WA were very similar. We noticed almost no quality loss ($< 0.1\%$) for both workloads. The other tested methods could achieve sometimes higher compression-ratio than our algorithm but with significant quality-loss. Keeping small quality-loss required to increase the sampling rate, but it decreased the compression-ratio. As expected, the results of random sampling based methods varied from run to run. In contrast, our method gave stable compression-ratio and quality. Changing the order of queries in the workload did not affect the quality loss and compression ratio by more than 5%.

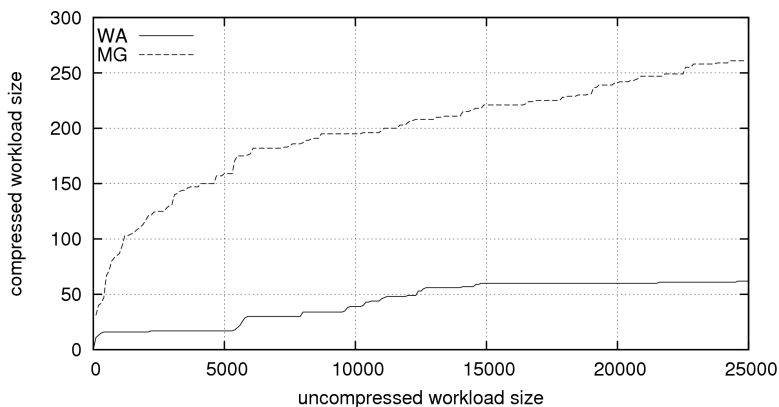


Fig. 3: Compressed workload size as a function of the input workload ($\delta = 0.66$)

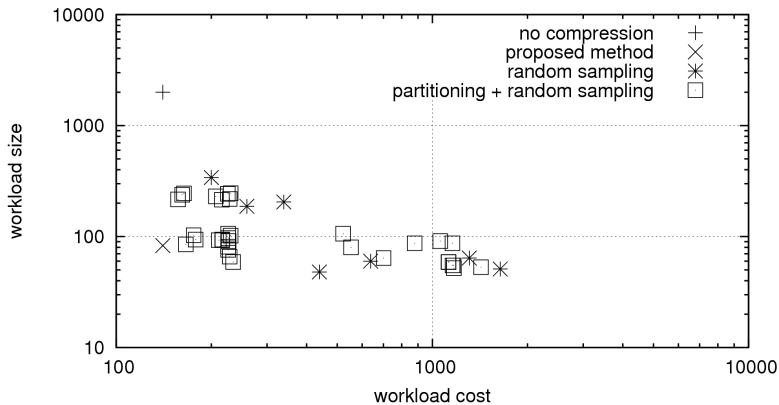


Fig. 4: Compression-ratio vs. ISP results quality for various algorithms for workload of 2000 queries executed by the application MG

6 Discussion

The experiments have shown that the presented compression method is useful for compressing transactional workloads and provides acceptable compression ratio and negligible quality loss of index selection results. This enables to deal with orders of magnitude less queries at the input of the index selection tool and to significantly reduce the database tuning time. Besides, the compression ratio can be easily adjusted by the parameter δ . However, it is not the best method when having the whole workload in advance. A k-Medoids variant like the one presented in [14] or some other standard clustering methods [16] can surely achieve better compression ratio within the same quality loss, because they pick the seed queries more carefully. This can lead to either smaller clusters or fewer of them. These algorithms are also usually not sensitive to the order of queries in the workload, unlike our method is.

7 Conclusions

We presented a simple solution of workload compression problem for large transactional database workloads. The main advantage of our method is the possibility to compress workloads incrementally, without storing all queries in memory, which makes our method ideal for usage with online database tuning software. The performed experiments have shown good compression ratio and low quality loss of the method, as opposed to random sampling based approach.

In the future we plan to investigate how the proposed compression method works for complex decision support workloads, e.g. TPC-H standard workload. We predict the compression ratio on such workloads would be worse than for the transactional workloads used in the presented experiments, because the decision support queries usually contain more predicates. Thus, the chances for

two queries to differ significantly in at least one predicate selectivity are greater. However, we cannot estimate how much this would affect the overall compression ratio and quality loss without actually measuring them. We are planning to do that as soon as the framework we have built for the predicate selectivity estimation supports subqueries and other structures usually used in the decision support queries e.g. `CASE WHEN`.

Acknowledgements We would like to thank Marzena Kryszkiewicz for important feedback on the method and experiments.

References

1. Finkelstein, S., Schkolnick, M., Tiberio, P.: Physical database design for relational databases. *ACM Trans. Database Syst.* **13**(1) (1988) 91–128
2. Ip, M.Y.L., Saxton, L.V., Raghavan, V.V.: On the selection of an optimal set of indexes. *IEEE Trans. Softw. Eng.* **9**(2) (1983) 135–143
3. Whang, K.Y.: Index selection in relational databases. In: *FODO*. (1985) 487–500
4. Barucci, E., Pinzani, R., Sprugnoli, R.: Optimal selection of secondary indexes. *IEEE Trans. Softw. Eng.* **16**(1) (1990) 32–38
5. Choenni, S., Blanken, H.M., Chang, T.: Index selection in relational databases. In: *International Conference on Computing and Information*. (1993) 491–496
6. Chaudhuri, S., Narasayya, V.R.: An efficient cost-driven index selection tool for Microsoft SQL Server. In: *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (1997) 146–155
7. Valentin, G., Zulliani, M., Zilio, D.C., Lohman, G., Skelley, A.: DB2 advisor: An optimizer smart enough to recommend its own indexes. In: *ICDE '00: Proceedings of the 16th International Conference on Data Engineering*, Washington, DC, USA, IEEE Computer Society (2000) 101
8. Zilio, D.C., Zuzarte, C., Lohman, G.M., Pirahesh, H., Gryz, J., Alton, E., Liang, D., Valentin, G.: Recommending materialized views and indexes with IBM DB2 design advisor. In: *ICAC '04: Proceedings of the First International Conference on Autonomic Computing*, Washington, DC, USA, IEEE Computer Society (2004) 180–188
9. Elnaffar, S., Powley, W., Benoit, D., Martin, P.: Today's DBMSs: How autonomic are they? In: *DEXA '03: Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, Washington, DC, USA, IEEE Computer Society (2003) 651
10. Ganek, A.G., Corbi, T.A.: The dawning of the autonomic computing era. *IBM Syst. J.* **42**(1) (2003) 5–18
11. Sattler, K.U., Schallehn, E., Geist, I.: Autonomous query-driven index tuning. In: *IDEAS '04: Proceedings of the International Database Engineering and Applications Symposium (IDEAS'04)*, Washington, DC, USA, IEEE Computer Society (2004) 439–448
12. Schnaitter, K., Abiteboul, S., Milo, T., Polyzotis, N.: Colt: continuous on-line tuning. In: *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, New York, NY, USA, ACM Press (2006) 793–795

13. Schnaitter, K., Abiteboul, S., Milo, T., Polyzotis, N.: On-line index selection for shifting workloads. In: ICDE Workshops, IEEE Computer Society (2007) 459–468
14. Chaudhuri, S., Gupta, A.K., Narasayya, V.: Compressing sql workloads. In: SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM (2002) 488–499
15. Garcia-Molina, H., Widom, J., Ullman, J.D.: Database System Implementation. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1999)
16. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann (2000)