A Decomposition Approach for Risk-Averse Index Selection

Rainer Schlosser
Hasso Plattner Institute, Germany
rainer.schlosser@hpi.de

Stefan Halfpap
Hasso Plattner Institute, Germany
stefan.halfpap@hpi.de

ABSTRACT

While finding the best selection of indexes is an important task the problem appears highly challenging as (i) indexes mutually affect their impact on performance and (ii) the number of index combinations can be enormous. Current selection approaches have limitations when problems are large and ignore the fact that future workloads are partially stochastic. In this paper, we propose a solver-based approach to find effective index selections for large-scale workloads. Our decomposition concept allows to deal with large candidate sets and makes it possible to address risk-averse problem versions, where multiple potential future workloads are taken into account. We demonstrate the applicability and the effectiveness of our approach for the TPC-DS benchmark workload. Our numerical results show that compared to state-of-the-art LP approaches index selections can be computed orders of magnitudes faster while still obtaining near-optimal performance results.

CCS CONCEPTS

• **Information systems** → *Database design and models*; *Data access methods*; • **Mathematics of computing** → *Linear programming*; *Integer programming*.

KEYWORDS

index selection, risk aversion, stochastic workloads, integer programming, mean-variance optimization

ACM Reference Format:
Rainer Schlosser and Stefan Halfpap. 2020. A Decomposition Approach for Risk-Averse Index Selection. In *32nd International Conference on Scientific and Statistical Database Management (SSDBM 2020), July 7–9, 2020, Vienna, Austria.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3400903.3400909

1 INTRODUCTION

Indexes are indispensable for speeding up queries on large databases. Because workloads are becoming both more complex, comprising of thousands of query templates referencing thousands of attributes, and dynamic, i.e., they change over time, automatic and efficient index selection approaches are important.

However, index selection (or index tuning), i.e., choosing the set of indexes to minimize the workload costs while considering a set of constraints, is an NP-hard problem [9]. Index selection consists of two steps. First, potentially suitable index candidates must be determined. Second, a subset of these candidates must be selected.

To solve large index selection problems, greedy heuristics, e.g., [4, 12], have been proposed, which often select or reject index candidates one after another. The main problem of greedy index selection algorithms is index interaction, i.e., the additional benefit of a specific index may depend on the selection of another index. As a result, an early chosen index (because of its overall large benefit for all queries) may potentially lose its benefit completely for the final selection, when additional indexes are added with higher benefits for individual queries.

To counteract this issue, alternative approaches seek to jointly select all indexes in a single step, i.e., solving the complete index selection as a multi-dimensional mathematical optimization problem, e.g., [5, 7]. Of-the-shelf (integer) solvers are tuned for varying input characteristics to prune sub-optimal solutions early and, thus, find optimal solutions significantly faster than naive enumerations. Nevertheless, when having too many index candidates, solvers take too long to find optimal solutions to be practically applicable.

Looking at existing selection approaches, we see two important aspects that render them unsuitable in real-world settings: (i) index interaction and (ii) huge candidate sets. Most approaches do not explicitly take the effects of index interactions into account or prune potential index candidates (too) early [12, 13], thereby degrading the solution's quality. We seek to address both aspects:

- We propose a solver-based decomposition heuristic to efficiently compute near-optimal index selections without being forced to strongly limit the set of candidates in advance.
- Further, instead of a single workload, we consider multiple potential future workload scenarios and compute risk-aware index selections using mean-variance optimization.
- We evaluate the performance of our decomposition approach against [5] (CoPhy) for the TPC-DS benchmark.
- We show that our concepts yield both, a near-optimal performance and significantly reduced computation times.

2 INDEX SELECTION PROBLEM

In this section, we describe the index selection problem. First, we discuss its main challenges (Section 2.1). Based on this discussion, we mathematically specify the index selection problem (Section 2.2). Finally, we discuss state-of-the-art LP approaches (cf. [5]) to select indexes from a given set of candidates (Section 2.3).

2.1 Challenges

2.1.1 Index Candidate Selection. The number of (relevant) single-attribute indexes can be directly derived by the overall number of (used) attributes. Typically, relational database systems also use multi-attribute indexes. To choose which of the relevant multi-attribute indexes to consider as candidates is difficult, because their number, i.e., the number of used attribute *combinations* in

SSDBM '20, July 07–09, 2020, Vienna, Austria

the queries, usually exceeds the number of attributes by orders of magnitude [13] and their benefit cannot be easily derived.

2.1.2 Index Interaction. Many indexes *interact* with each other, i.e., the benefit of one index is affected by the presence of another [11]. Thus, to find effective selections, the mutual interplay of indexes has to be taken into account. However, index interaction increases the complexity of (i) choosing suitable candidates and (ii) computing their best selection significantly, as the benefit of an index cannot be expressed or ranked independently.

2.1.3 What-if Based Cost Estimations. A further challenge is an efficient cost estimation of the workload with indexes. Repeatedly creating a large set of index combinations physically and executing queries is too expensive. Hence, many existing approaches [4, 5] rely on the *what-if capability* of an optimizer, which allows estimating the size and benefit of so-called *hypothetical indexes* based on statistics without physically creating the indexes. Note, including the optimizer for the determination of query costs guarantees that indexes are really used during query execution [4]. Unfortunately, what-if calls require time-consuming query optimizations by the optimizer. Resulting, what-if calls may be the bottleneck for index selection approaches (cf. [8]). The number of what-if calls during the index selection can be reduced by avoiding unnecessary calls using caching and by exploiting workload context information. Further, one can speed-up what-if calls by INUM's advanced caching mechanism [8], exploiting that optimal query plans for different index sets often have the same structure and only differ in the table access costs.

2.1.4 Potential Future Workloads. Workload anticipations are crucial. Potential future workload scenarios can be determined, e.g., based on previously observed (seasonal) workloads or forecasts. Regarding a suitable index selection, such scenarios allow optimizing the expected performance (risk-neutral) or more robust (risk-averse) objectives. We assume that a workload scenario is characterized by a set of queries, which occur with given frequencies within a certain time frame. We consider K potential workload scenarios with probability P_k, $k = 1, ..., K$, where $\sum_k P_k = 1$.

2.2 Mathematical Problem Description

We consider the problem to choose secondary indexes for a workload, consisting of Q queries and N attributes, such that the (expected) overall performance is maximized, e.g., by minimizing the execution time. Each query j is characterized by a set of attributes $q_j \subseteq \{1, ..., N\}$, $j = 1, ..., Q$, that are accessed during query evaluation. A (multi-attribute) index i is characterized by an *ordered* set of attributes from $\{1, ..., N\}$. Further, by I, we denote a set of *index candidates* and by the subset $I^* \subseteq I$, we denote an *index selection*. Using binary variables x_i, we indicate whether a candidate index $i \in I$ is part of the selection I^*, i.e., $I^*(I, \vec{x}) := \bigcup_{i \in I : x_i = 1} \{i\}$. The costs for a query q_j, $j = 1, ..., Q$, are denoted by values $c_j(I^*)$, which depend on the index selection I^*. Typically the costs $c_j(I^*)$ are determined by what-if optimizer calls. Note, a query j can be of various type. The total workload costs C_k of scenario k are defined by the (weighted) sum of query costs c_j of all queries q_j, multiplied by their number of occurrences denoted by f_{jk}, $j = 1, ..., Q$, $k = 1, ..., K$, i.e.,

$$C_k(I^*) := \sum_{j=1,...,Q} f_{jk} \cdot c_j(I^*). \tag{1}$$

Further, we assume that the memory consumed by the selected indexes is not allowed to exceed a certain budget A. The necessary memory for a (multi-attribute) index i, $i \in I$, is denoted by m_i. The total memory used by a selection I^* amounts to

$$M(I^*) := \sum_{i \in I^*} m_i. \tag{2}$$

Finally, the index selection problem can be defined by:

$$\underset{x_i \in \{0,1\}, i \in I}{\text{minimize}} \quad \sum_{k=1,...,K} P_k \cdot C_k(I^*(I, \vec{x})) \tag{3}$$

$$\text{subject to} \quad M(I^*(I, \vec{x})) \leq A. \tag{4}$$

Note, for problem (3) - (4) *both* the selections $\vec{x}$ as well as the index candidate set I determine the solution quality.

2.3 Linear Programming Approach

In this subsection, we discuss existing LP approaches to solve index selection problems. In this context, we resemble the concept of CoPhy [5], abstracting from multiple query plans per query with slots for accessing tables. Equal to CoPhy, which allows applying a single index per slot, we allow a single index per (sub) query. As a result, the structure and complexity of both linear programming models are the same and depend crucially on the number of options, via variables and constraints. For ease of simplicity, we do not consider updates.

Moreover, we assume that the costs of (a fixed query plan and) a query when using a specific index are at hand (e.g., based on what-if calls), i.e., if an index i is applied to query j, we consider scan costs $c_j(i)$, $i \in I \cup 0$, where 0 describes the option that no index is applied to j, $j = 1, ..., Q$. Besides x_i, we use the additional binary variables z_{ji} to model whether an index option i is applied to a query j, which particularly depends on the selection of other indexes that might be more beneficial (cf. index interaction).

Further, by $I_j \subseteq I$, we denote the (sub)set of index candidates out of I that are applicable to query j, $j = 1, ..., Q$. For a given candidate set I and K potential workload scenarios, the basic structure of index selection LP approaches (cp. CoPhy) minimizing (expected) total costs can be written as:

$$\underset{\substack{z_{js}, x_i \in \{0,1\}, i \in I, \\ j=1,...,Q, s \in I_j \cup 0}}{\textit{minimize}} \quad \sum_{k=1,...,K} P_k \cdot \sum_{\substack{j=1,...,Q, \\ i \in I_j \cup 0}} f_{jk} \cdot c_j(i) \cdot z_{ji} \tag{5}$$

$$\text{subject to} \quad \sum_{i \in I_j \cup 0} z_{ji} = 1 \quad \forall j = 1, ..., Q \tag{6}$$

$$z_{ji} \leq x_i \quad \forall j = 1, ..., Q, i \in I_j \tag{7}$$

$$\sum_{i \in I} m_i \cdot x_i \leq A. \tag{8}$$

The family of constraints (6) guarantees that a unique index option is used for each query j. The constraints (7) serve to identify which indexes i are used at all. Constraint (8) ensures that the memory budget A is not exceeded, cf. (4). The linear programming formulations, cf. (5) - (8), require all cost coefficients $c_j(i)$, e.g., determined by what-if optimizer calls. Note, that what-if cost estimations can be efficiently and accurately derived, e.g., via INUM [8].

Remark 1 *Approaches like (5) - (8) are superior to rule-based heuristics but do not scale as the problem complexity strongly increases in the number of queries Q and candidates $|I|$. Hence, such solver-based approaches are (i) either not applicable if the problem is sufficiently large, see [10] (Table I), or (ii) lead to suboptimal selections using reduced candidate sets, see [10] (Figure 3 - 4).*

3 SCALABLE DECOMPOSITION HEURISTIC

In the following, we present our heuristic decomposition approach to solve large-scale (risk-neutral) index selection problems (Section 3.1). The approach can also be applied to more complex risk-averse problem versions (Section 3.2).

3.1 Decomposition Approach

To circumvent the fact that the LP (5) - (8) does not scale, next, we propose an approach to decompose the index candidate set to be able to reduce the LP's complexity in a targeted way.

Algorithm 1 (Problem Decomposition)

Step 1: Partition the set of index candidates I into B smaller subsets. The number B, the chunk sizes, and the subsets can be arbitrarily defined. For example, subsets of (at most) size S can be formed in a naive way. Then the number of chunks is $B := \lceil |I|/S \rceil$.

Step 2: For all subsets of index candidates $\tilde{I}_b$, $b = 1, ..., B$, we solve the problem (5) - (8) independently (with the same budget A) and store the selected indexes, i.e., the sets $\tilde{I}^*_b$, $b = 1, ..., B$.

Step 3: Solve the problem (5) - (8) for the combined set of remaining index candidates $\tilde{I} := \bigcup_{b=1,...,B} \tilde{I}^*_b$.

The key idea of Step 1 and Step 2 is (i) to solve LPs of smaller size and (ii) to exclude indexes which are dominated by others. In Step 3 the best selection of the best indexes of the individual subsets are formed. The subproblems' complexity can be controlled by choosing the chunks' sizes. In case the candidate set $\tilde{I}$ (after Step 2) is too large, Step 1-2 can be repeatedly applied (to $\tilde{I}$) or Step 2 can be executed with a smaller budget $\tilde{A} < A$. Moreover, to optimize the choice of chunks, further information can be used, e.g., by collecting *similar* indexes within chunks. This way index interaction is more effectively addressed in each chunk.

3.2 Mean-Variance Optimization

In this section, we combine our decomposition approach with more complex non-linear problem versions, which address risk-averse selections. An effective criteria to balance expected performance vs. performance deviations is mean-variance optimization (MVO). Based on a scenario k's workload costs $C_k(\vec{z}) = \sum_{j=1,...,Q, i \in I} f_{jk} \cdot c_j(i) \cdot z_{ji}$, $k = 1, ..., K$, cf. (1) & (5), the *expected workload costs* of an index selection (determined by $\vec{z}$) are given by

$$EC(\vec{z}) = \sum\nolimits_{k=1,...,K} P_k \cdot C_k(\vec{z}) \tag{9}$$

and the associated *variance* of workload costs amounts to

$$VC(\vec{z}) = \sum\nolimits_{k=1,...,K} P_k \cdot (C_k(\vec{z}) - EC(\vec{z}))^2. \tag{10}$$

The trade-off between expected costs EC and the variance VC can be modeled by the extended objective, cp. (5),

$$\text{minimize}_{\vec{x},\vec{z}} \quad MVO(\vec{z}) = EC(\vec{z}) + \alpha \cdot VC(\vec{z}) \tag{11}$$

using a penalty parameter $\alpha \geq 0$. The penalty term in the objective provides an incentive to *avoid large deviations* in potential workload costs by selecting indexes such that queries of comparably heavy workload scenarios are sped up. The decision variables and the constraints (6) - (8) remain *unchanged*. While for α=0 we obtain the linear risk-neutral model (5) - (8), for α>0 we have a (more complex) binary quadratic problem (BQP), which can be solved using standard solvers as long as the problem is sufficiently small.

Table 1: Performance comparison: Workload costs EC, cf. (9), and runtimes of Algorithm 1 for different numbers of chunks B with size S vs. the optimal solution (EC^*) of the LP model (5) - (8) with full $S = |I|$, cf. CoPhy's approach; TPC-DS with $|I| = 8\,343$ index candidates (1-3 attributes).

B	Size S	$\lvert\tilde{I}\rvert$	EC/EC^*	Time (Step 2 + 3)	time saved
1	8 343	8 343	(100.00%)	3.5 s	(100%)
4	2 100	474	100.00%	2.0 s + 0.1 s = 2.1 s	-46%
16	525	586	99.98%	1.4 s + 0.2 s = 1.6 s	-54%
42	200	784	100.00%	3.6 s + 0.3 s = 3.9 s	+11%

4 NUMERICAL EVALUATION

In this section, we compare the performance of Algorithm 1 against optimal selection approaches with *full* candidate sets (cf. $B = 1$). Using the setup described in Section 4.1, we evaluate the risk-neutral (Section 4.2) as well as the MVO model (Section 4.3).

4.1 Setup and Model Input

To obtain the model inputs, we set up a PostgreSQL 11 [3] database system with the extension HypoPG [2]. HypoPG allows creating hypothetical indexes and estimating their sizes. We loaded TPC-DS tables with scale factor 1. For the TPC-DS queries ($Q = 99$) and various index configurations I, we derived query costs $c_j(i)$, $i \in I$, by calling PostgreSQL's `EXPLAIN` with the query template string using fixed parameters. For all queries, we use $E(f_{jk}) := 1, j = 1, ..., Q, k = 1, ..., K$, to obtain, on average, an identical number for each query template. Note, to avoid a skew in the queries' workload shares or in the total workloads of single scenarios, the frequencies can also be normalized accordingly. Index sizes are modeled by using HypoPG's function `hypopg_relation_size()`. Obtained model inputs to reproduce the calculations are available online [1].

4.2 Risk-Neutral Performance Comparison

We compare our decomposition-based approach (cf. Section 3.1) with the solutions of the state-of-the-art approach CoPhy numerically, focussing on the workload costs and the required runtimes. Table 1 shows results for the TPC-DS benchmark workload. We let K=10 and use randomized query frequencies $f_{j,k}$:=Uniform(0,2), $k = 1, ..., K$. The budget is $A := 0.0002 \cdot M(I)$, cf. (2).

We observe that while the solution performance (EC/EC^*), cf. (5) & (9), of Algorithm 1 is constantly (near-)optimal for different chunkings, the number of chunks B majorly affects the runtime, which is dominated by Step 2. The u-shaped dependence of the runtime can be explained as follows: First, an increase in B *decreases* the total runtime due to the reduced problem complexity. Then, at some point (cf. $B > 16$) Step 2's runtime *increases* again as (i) the reduced complexity, cf. S, does not compensate the increased number of subproblems B anymore and (ii) during Step 2 more indexes survive, cf. $|\tilde{I}|$, such that Step 3 takes longer. Note, for the extreme case $S = 1$, in Step 3, we again obtain the original problem. Further experiments show that the results obtained also hold for various other memory budgets A.

Remark 2 *In Algorithm 1, the selection within each chunk (Step 2) eliminates candidates that are dominated by others. Beneficial indexes are hardly excluded, since (compared to the original problem) within*

Table 2: Performance comparison: Workload costs and runtimes of the optimal BQP, cf. (11), vs. our decomposition heuristic (Algorithm 1) for different chunk sizes S; TPC-DS with $|I| = 1\,422$ index candidates (1-2 attributes).

B	Size S	$\lvert\tilde{I}\rvert$	MVO/MVO^*	Time (Step 2 + 3)	time saved
1	1 422	1 422	(100.00%)	172 s	(100%)
4	400	24	100.00%	21.5 s + 0.5 s = 22 s	-87%
15	100	124	100.00%	4 s + 3 s = 7 s	-96%
57	25	421	100.00%	9 s + 5 s = 14 s	-92%

Table 3: Performance comparison: Workload costs and runtimes of the optimal BQP, cf. (11), vs. our decomposition heuristic (Algorithm 1) for different chunk sizes S; TPC-DS with $|I| = 8\,343$ index candidates (1-3 attributes).

B	Size S	$\lvert\tilde{I}\rvert$	MVO/MVO^*	Time (Step 2 + 3)	time saved
1	8 343	8 343	no result	no result	no result
4	2 100	140	("100.00%")	1 950 s + 5 s = 1 955 s	(100%)
8	1 050	59	"100.00%"	267.5 s + 0.5 s = 268 s	-86%
16	525	142	"100.00%"	149 s + 6 s = 155 s	-92%
42	200	458	"100.00%"	248 s + 43 s = 291 s	-85%

*a chunk the competition between indexes (cf. index interaction) is **lower** as less indexes are involved and the budget is unchanged. In the final selection (Step 3) all non-dominated indexes compete against each other in a joint setting (with the highest degree of competition), which finally provides near-optimal results.*

4.3 Risk-Averse Performance Comparison

Next, we study the quality of Algorithm 1 applied in the risk-averse MVO model (cf. Section 3.2). We consider the parameter setting as used in the example of Table 1. The penalty factor is $\alpha = 1$, cf. (11). To solve the BQP, we used the Gurobi solver (Version 8.1.0).

Using only indexes with 1-2 attributes (in total 1 422), in the risk-averse MVO case, see Table 2, (for different S) we obtain the best index selection with the optimal objective value (cf. B=1). The runtime is minimized for S=100 (B=15). Further, the saved runtime is significantly higher, which is due to the higher complexity of the initial problem. Again, for different budgets, we find that the MVO performance is not affected by S, cf. Remark 2.

Considering all indexes with 1-3 attributes (in total 8 434), see Table 3, the *optimal* BQP solution with a full candidate set could *not* be computed anymore, which shows the weakness of models with large candidate sets, cf. Remark 1. In contrast, our decomposition approach still allows computing high quality results in a reasonable amount of time. Note, in Table 3, the results for $B = 4$ chunks serve as the reference result with "100%" performance. Recall, in general, if the set of candidates used contains more indexes (cf. $|I|$=1 422 vs. 8 343), the solution is better but also harder to compute.

Remark 3 *We find that Algorithm 1 can effectively improve solver-based approaches, which have limitations when candidate sets are chosen too large (high complexity) or too small (exclusion of beneficial indexes). We recommend choosing the chunking in Algorithm 1 such that Step 2's subproblems best benefit from a smaller candidate set, i.e., a suitable number of chunks is exceeded if Step 2's total runtime does not decrease in B anymore (i.e., in our example $B = 16$).*

5 RELATED WORK

Early approaches tried to derive optimal index configurations by evaluating attribute access statistics [6]. Newer index selection approaches are coupled on the query optimizer of the database system. By doing so, the costs models of the index selection algorithm and the optimizer are the same. As a result, the benefit of considered indexes can be estimated consistently [4]. As optimizer invocations are costly, especially for complex queries, along with improved index selection algorithms, techniques to reduce and speed up optimizer calls have been developed [4, 8].

An increasing number of possible optimizer calls for index selection algorithms opens the possibility to investigate an increasing number of index candidates. Compared to greedy algorithms [4, 12], approaches using mathematical optimization, especially integer linear programming [5], are able to efficiently evaluate a larger number of index combinations. A major challenge of these index selection approaches is dealing with the complexity of integer programming, which is in general not scalable. An obvious solution is reducing the number of considered index candidates. Instead of restricting the set of index candidates in advance, we propose a decomposition approach to handle the complexity.

6 CONCLUSIONS

We have proposed a solver-based decomposition approach, which allows computing near-optimal index selections for problems with large candidate sets. For the TPC-DS workload, we verified the near-optimal performance of our approach against optimal solutions, while the required runtime can be effectively reduced. With our approach index candidate sets do not have to be strongly limited in advance, which is beneficial as (i) potentially suitable indexes are not excluded and (ii) heuristic preparations of candidate sets can be omitted. Moreover, we showed that our decomposition concept can also be used to identify robust index selections in the presence of multiple potential future workloads.

REFERENCES

[1] [n.d.]. https://hyrise.github.io/replication/.
[2] HypoPG. https://github.com/HypoPG/hypopg.
[3] PostgreSQL. https://www.postgresql.org.
[4] Surajit Chaudhuri and Vivek R. Narasayya. 1997. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In *Proc. VLDB'97*. 146–155.
[5] Debabrata Dash, Neoklis Polyzotis, and Anastasia Ailamaki. 2011. CoPhy: A Scalable, Portable, and Interactive Index Advisor for Large Workloads. *PVLDB* 4, 6 (2011), 362–372.
[6] Sheldon J. Finkelstein, Mario Schkolnick, and Paolo Tiberio. 1988. Physical Database Design for Relational Databases. *ACM Trans. Database Syst.* 13, 1 (1988), 91–128. https://doi.org/10.1145/42201.42205
[7] Stratos Papadomanolakis and Anastassia Ailamaki. 2007. An Integer Linear Programming Approach to Database Design. In *Workshop@ICDE*. 442–449.
[8] Stratos Papadomanolakis, Debabrata Dash, and Anastassia Ailamaki. 2007. Efficient Use of the Query Optimizer for Automated Database Design. In *Proc. VLDB 2007*. 1093–1104.
[9] Gregory Piatetsky-Shapiro. 1983. The Optimal Selection of Secondary Indices is NP-Complete. *SIGMOD Record* 13, 2 (1983), 72–75.
[10] Rainer Schlosser, Jan Kossmann, and Martin Boissier. 2019. Efficient Scalable Multi-attribute Index Selection Using Recursive Strategies. 1238–1249.
[11] Karl Schnaitter, Neoklis Polyzotis, and Lise Getoor. 2009. Index Interactions in Physical Design Tuning: Modeling, Analysis, and Applications. *PVLDB* 2, 1 (2009), 1234–1245.
[12] Gary Valentin, Michael Zuliani, Daniel C. Zilio, Guy M. Lohman, and Alan Skelley. 2000. DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes. In *Proc. ICDE*. 101–110.
[13] Daniel C. Zilio, Jun Rao, Sam Lightstone, Guy M. Lohman, Adam J. Storm, Christian Garcia-Arellano, and Scott Fadden. 2004. DB2 Design Advisor: Integrated Automatic Physical Database Design. In *Proc. VLDB*. 1087–1097.