Automated Clustering Recommendation With Database Zone Maps

Suratna Budalakoti
Oracle Corporation
Redwood Shores, CA, USA
suratna.budalakoti@oracle.com

Mohamed Ziauddin
Oracle Corporation
Redwood Shores, CA, USA
mohamed.ziauddin@oracle.com

Andrew Witkowski
Oracle Corporation
Redwood Shores, CA, USA
andrew.witkowski@oracle.com

You Jung Kim
Oracle Corporation
Redwood Shores, CA, USA
you.jung.kim@oracle.com

Ramarajan Krishnamachari
Oracle Corporation
Redwood Shores, CA, USA
ramarajan.krishnamachari@oracle.com

Alan Wood
Oracle Labs
Redwood Shores, CA, USA
alanw@ieee.org

ABSTRACT

Zone maps are a data access structure provided in multiple data warehousing products [1, 3, 5, 14, 20], that reduce the cost of data access for a query by advance pruning of irrelevant data blocks. This is achieved by dividing the table into zones and storing for each zone, the minimum and maximum value of selected columns. The structures used to store these values are called zone maps. Zone maps are most effective for a query if the data is sorted (or clustered) on a column that the query has filters on, and if it also has zone maps constructed on the column. More selective filters are likely to lead to better performance. But as filter predicates vary from query to query, it is not straightforward to identify a single clustering that is effective across the entire workload. This paper describes a novel method for automatically analyzing a workload and recommending a clustering solution, as well as the zone maps to construct for improved performance. It works for both linear and interleaved (z-order) clustering, the two most commonly used clustering methodologies. In our experiments on the TPCH benchmark and customer datasets, this approach provides significant performance improvements over the baseline. To the best of our knowledge, this is the first work to address the automated clustering (auto-clustering) problem. The method described here has been implemented and will be released in an upcoming version of the Oracle Autonomous Database.

CCS CONCEPTS

• **Theory of computation → Database query processing and optimization (theory)**.

KEYWORDS

auto-clustering, clustering, databases, zone maps, z-order

ACM Reference Format:
Suratna Budalakoti, Mohamed Ziauddin, Andrew Witkowski, You Jung Kim, Ramarajan Krishnamachari, and Alan Wood. 2024. Automated Clustering Recommendation With Database Zone Maps. In *Companion of the 2024 International Conference on Management of Data (SIGMOD-Companion '24), June 9–15, 2024, Santiago, AA, Chile.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3626246.3653397

SIGMOD-Companion '24, June 9–15, 2024, Santiago, AA, Chile

ACM ISBN 979-8-4007-0422-2/24/06.
https://doi.org/10.1145/3626246.3653397

1 INTRODUCTION

Zone maps are data access structures that allow the database to prune data blocks containing rows irrelevant to a query predicate by using previously extracted and stored aggregate information (specifically, minimum/maximum values). The pruned database blocks need not be accessed during query execution, which improves query performance. To construct a zone map, the entire table is divided into equally-sized groups of blocks, called zones. Then for each column or expression of interest, the minimum and maximum value per zone is stored in a data structure. For an equality or range predicate, a zone can be ignored if the predicate values have no overlap with the minimum to maximum range of the zone. The data structure containing the minimum and maximum value of each zone is called a zone map.

Zone maps work best for equality and range predicate queries, provided the values in a zone are concentrated in a small subset of the column's range: for example, if the data was sorted by the column. As a simple example, consider a column consisting of US states in a table. If the table is sorted on this column, we can imagine the minimum and maximum value of the first zone could be 'AK' and 'AZ' respectively. Given a zone map on this column, only the first zone needs to be accessed to answer a query with an equality predicate of State = 'AK', and all the other data blocks can be pruned. In comparison, if the table is not sorted on this column, the minimum and maximum value of each zone could include 'AK' in its range, and the database might potentially have to access all zones. So, a key challenge in using zone maps is identifying the right columns to sort the table on.

Zone maps, however, are not the only access structure that benefit from data clustering. For example, Oracle Exadata systems maintain *storage indexes* [13] for columns, which store minimum/-maximum values per 1 Megabyte region, and prune unnecessary blocks. Another benefit of data clustering is that it leads to better data compression. All these factors in combination hold the promise of significant performance improvement, given an appropriate clustering.

Also, a complete sort of the data is not required. As long as the values in a zone are within a particular range, they do not

have to be sorted *within* the zone. To make this distinction, the term *clustered* is generally used in the context of zone maps, to identify columns where all values within a zone are within a certain minimum and maximum value. However at present, most databases perform clustering by sorting the column, so there is no practical difference between the two terms. In this paper, the terms clustering and sorting are used interchangeably.

Most database tables, however, have multiple columns that are important to queries, so clustering on a single column is usually not enough. To address this, databases may provide options for combining multiple columns into a single clustering. Two common options are *linear clustering*[6, 20] and *interleaved clustering* [11, 15, 20], also known as *z-order clustering*. Both of these clustering methods are supported by the Oracle database, and also by other database products. Both of these options are discussed in Section 2.

Also, in the context of data warehouses where zone maps are generally used, the Fact tables are the largest tables where fast data access is crucial. At the same time, there are often a large number of filter predicates on Dimension columns (usually the Fact table is joined with the Dimension table on a join column, before the filter predicate is applied). As a result, the most effective use of zone maps might be to construct them on Dimension table columns. To address this, zone maps are sometimes constructed by first temporarily joining the Fact table with the Dimension table, and then storing the min/max values for the relevant Dimension table columns in a data structure.

Given a workload, this paper describes an efficient approach to automatically identify the columns to cluster on, and the columns to maintain zone maps for. The approach achieves efficiency by relying on two key techniques. First, it builds a probabilistic model of the workload queries by extracting characteristics of the queries that are relevant to zone map construction, such as frequency of each column's usage in queries. Following this, the query workload model is used as input to a probabilistic model, which is capable of extrapolating performance on the workload from performance on a sample of rows taken from the database tables. The performance estimates provided by the probabilistic model are in turn used by a search algorithm to search through the space of possible solutions. The methodology described here has already been implemented in the Oracle RDBMS for release.

The paper is organized as follows: Section 2 describes the problem background, including the two common clustering methods: linear and interleaved clustering. It also provides intuition for what qualities a good clustering solution should have. Section 3 discusses the Related Work. Sections 4 and 5 discuss the approach architecture, with its four main components: the Query Workload Modeler, the Clustering Quality Evaluator, the Solution Search Algorithm, and the Zone Maps Recommender, and how they are connected to each other. Sections 6-10 discuss the above four components in detail. Section 11 discusses database verification, a step taken to verify clustering performance before it is implemented in a customer database. Following this, Section 12 presents the results of experiments on the TPCH benchmark along with two customer workloads, and Section 13 concludes the paper.

2 BACKGROUND AND PROBLEM DESCRIPTION

For a database workload (the term workload is used here to refer to a database instance populated with data, along with a set of representative queries on the data), this paper presents a method to identify which columns to cluster on, as well as, which columns to build zone maps on. The approach first identifies the best columns to cluster on, and then identifies any additional columns that would benefit from a zone map (apart from the ones chosen for clustering). The automated clustering algorithms described by this paper are applicable to and evaluated on both linear and interleaved clustering. The next two sub-sections describe linear and interleaved clustering respectively. Following this, Section 2.3 discusses the intuition behind what constitutes a good clustering.

2.1 Linear Clustering

The database performs a linear clustering on multiple columns by performing an order-by sort on the given columns. To be more precise, given a linear clustering Linear(T.A,T.B,T.C) on columns A,B,C of table T, the RDBMS proceeds as follows: first the entire table is sorted on column A, followed by a sort on column B for keys that are tied in the values of column A, and then on column C for ties on A, B. In such cases, the system can expect the best performance for queries with predicates on column A, followed by queries on column B, and then queries on column C. As a quick example, consider a database table with two columns, State and Zip Code, with the following three rows: (TX,78712), (CA,94404), (CA,94401). Then the linear sorted order would be: (CA,94401), (CA,94404), (TX,78712). This is because the rows were first sorted by State, and then the two tied rows (where State = 'CA') were sorted by Zip Code.

In general, hierarchical or many-to-one relationships, such as those between State and Zip Code, are a good choice for linear clustering. This is because clustering by State also clusters Zip Code to a large extent. In the absence of such relationships, the clustering gains for second and later columns in a linear clustering are much smaller than those for the first column, but may still be worthwhile if incremental gains are significant enough. Linear clustering is also a good choice if a particular column is key to workload performance, and interleaving it with other columns will lead to performance degradation.

2.2 Interleaved Clustering

To perform an interleaved clustering Interleaved(T.A,T.B) on columns A,B of table T, the RDBMS proceeds as follows: a temporary virtual column is created which does a bitwise interleaving of the values in columns A and B for each row, and the table is sorted on this temporary column. Bitwise interleaving, also known as *z-order curve* [17, 18] is a common technique for ordering multi-dimensional data in a single dimension, while preserving a certain degree of data locality. In the case of database clustering, it creates an ordering where for each unique combination of the interleaved columns, all occurrences are located adjacent to each other in a single *location* (we use the word location to refer to a contiguous sequence of data blocks). This provides a high degree of locality that zone maps can benefit from. Apart from this, for each individual column that is

part of the clustering, interleaved clustering will impose a certain degree of data locality, even though all keys will not be at a single location.

As an example, consider a database table in a retail store containing customer orders, which has been clustered (interleaved) on columns Month and Item_Id. This clustering will be highly efficient if most queries are searching for information about certain Item_Ids in certain months, as all rows with a particular (Item_Id, Month) combination will be located adjacent to each other in a single location. The database system can use zone maps information to ensure it accesses only these data blocks. However, for queries that are looking for all rows with a specific Item_Id (say Item_Id 100), rows with a particular Item_Id would no longer be clustered adjacently in a single location. Instead the database system might have to visit up to twelve locations ((100, January), (100, February), and so on) to find all rows where $Item\_Id = 100$. This is, however, the worst-case scenario. For example, if the item is sold only in a few months (for example, Christmas trees in December), such locations might be far fewer than twelve. It is also possible that some queries in the workload have highly selective filters on the Item_Id column, while others have selective filters on the Month column. In such cases, an interleaved clustering might be able to provide reasonable performance for both these sets of queries.

Also, in certain special cases, such as many-to-one mapping, an interleaved clustering may be as efficient as linear clustering on one column. As another example, if we have an interleaved clustering on columns Zip Code and State, since there is a many-to-one mapping from Zip Code to State, all rows with a particular Zip Code would still be clustered into one location. Note, however, that in this case the keys for each State (for example, 'CA') would be distributed to multiple locations.

In general, interleaved clustering is more flexible, as it can provide reasonable performance across a range of correlations between columns (from non-correlated to mildly correlated to hierarchical relationships). Interleaved clustering also does a better job of balancing different query types in a complex workload, where different subsets of the query workload has selective filters on different sets of columns.

2.3 Clustering Intuition

Apart from the differences between linear and interleaved clustering, there are several factors that impact whether a clustering on a column would be useful:

(1) *Column Importance and Selectivity*: Only columns that feature in a significant portion of filter predicates in the query workload should be considered. Apart from this, it is better to cluster on columns for which queries are highly selective, because if queries on a column usually select a majority of rows, all zones will need to be accessed irrespective of the existence of zone maps.
(2) *Column Heterogeneity*: We refer to the distribution of the number of terms in 'OR' filter predicates on a column as the column's heterogeneity (for example, the filter predicate "State = 'CA' Or State = 'TX'" has a heterogeneity of 2). The larger the number of 'OR'-operator terms for a query, or the larger the in-list[1], the higher its heterogeneity. Higher heterogeneity leads to worse zone maps performance, because a zone will need to be accessed if even one value on an in-list is present in the zone.
(3) *Relationship with Other Columns*: The relationship between columns has a crucial impact on the quality of a clustering solution. Given below are some examples of the ways in which the relationship between columns is relevant to clustering.
   (a) Consider two columns, US Zip Code and US State. These columns are not statistically independent. Even further, due to the way Zip Code and State are organized in the US (where Zip Code for a given State all start with the same digit), if the data is clustered on Zip Code, it is automatically clustered on State. As a result, clustering on one column (Zip Code) gives us useful zone maps on two columns (Zip Code and State), without having to cluster on State. This makes Zip Code a good candidate for a clustering column.
   (b) Consider two statistically independent columns, State and Employee Name. Linear clustering on only one of these columns will not be sufficient, as a zone map on Employee Name, created after clustering on State, is unlikely to contain names restricted to a small subset of the A-Z range. Instead, a two-column linear clustering would be required. Also, since names can be close to unique, linearly clustering on 'Employee Name, State' would essentially leave 'State' unclustered, as there will be very few ties between names. In comparison, clustering on 'State,Employee Name' is likely to give better performance. An alternate choice in this case might be to use an interleaved clustering, as it is more robust with respect to choice of the first clustering column.

Note, however, that due to the complicated nature of across-column dependencies, this approach does not try to explicitly model these dependencies. Instead, zone maps are constructed on a sample for different candidate clustering solutions, and the min/max values of these sample-based zone maps are used to estimate a candidate solution's performance. The min/max values for each sample zone implicitly include the effect of these interactions.

3 RELATED WORK

To the best of our knowledge, this is the first work to address the problem of automated clustering (or sorting order) recommendation for zone maps. The closest related work is in the field of *instance-optimized data layout* [4] techniques. These methods custom-fit the data layout to a specific workload, using highly specific information such as the exact filter predicates used in the workload. This might be done by constructing a binary decision tree structure such as a qd-tree [19], which assigns each row in a table to a particular block. At query execution time, the same qd-tree is used to determine which blocks to access. While instance-optimized approaches are likely to outperform straightforward clustering/sorting on most workloads, as they are tailored specifically to the workload filters, we believe auto-clustering provides other advantages that outweigh the performance gains for our situation. These advantages are listed

[1] https://docs.oracle.com/cd/B19306_01/server.102/b14200/conditions013.htm

next. Apart from mitigating the usual risks of overfitting such as data/workload changes, auto-clustering solutions are more intuitive and easily understood by users, as database administrators are used to sorting data on columns for performance. This makes them more likely to be accepted. Apart from this, clustering solutions have low development and resource costs, as sorting functionality is available and efficient on most databases. For these reasons, we decided to investigate auto-clustering solutions in this work.

In the industrial domain, currently clustering columns and zone maps in the Oracle database (and other databases) are usually selected via human input. In some databases, the system is able to make suggestions based on heuristics: for example, date columns are often suggested as a good candidate for zone maps in data warehousing solutions, since typically data is loaded in time intervals. An alternate approach is used by the Snowflake database [2]. While the Snowflake database does not provide automated clustering recommendations[3], it provides users with access to heuristic measures of clustering quality such as overlap and clustering depth [8], so that they can manually compare candidate solutions. The heuristic measures, however, do not consider workload characteristics such as individual column importance, column selectivity and *heterogeneity* (defined in Section 6.2), and it is not clear if such heuristics can be extended to include these. These aspects of the evaluation are instead left to the user. To help with this, Snowflake provides detailed documentation with guidance on selecting good clustering candidates, such as using columns "that are most actively used in selective filters"[9]. In contrast, by using a more statistically grounded model of clustering quality, combined with a query workload model, our approach is able to automatically compare candidate solutions while considering both data and query characteristics. This allows us to make automatic clustering recommendations without user input.

Another alternate approach, similar to clustering because it reorganizes database tables to align them with query filters, is Database Cracking [10]. Like clustering, cracking orders (by partially sorting) the data within each database column to align it with workload characteristics. However, the specifics of the goal and methodology for database cracking are different. The goal of cracking is to identify regions *within* each column that require ordering (so that only the frequently used keys in the column are ordered, and the rest are not moved, saving on cost). Also, unlike automated clustering, cracking does not address the problem of relationships between columns. To work around the impact of these relationships, it stores and re-orders each column independently: so it is a better fit for column-oriented databases. Besides this, cracking does not explicitly model the query workload, or organize the data in advance. Instead the data is organized in real-time, in response to each incoming query (though this is a design choice, and it may be possible to remove this limitation).

[2]The Snowflake database does not use zone maps. Instead it divides each database table into a large number of micro-partitions. Summary statistics (such as min/max values) are maintained for each micro-partition, which allow for data pruning in a way similar to zone maps. As a result, just like zone maps, clustering on the right set of columns is key to performance for the Snowflake database.

[3]While the Snowflake database has an 'automatic clustering' feature [7], this features refers to automatic re-clustering of tables where columns are no longer clustered due to DML (inserts/updates, etc). Clustering columns are assumed to have already been defined by the user.

4 APPROACH ARCHITECTURE

In theory, we could identify a good clustering solution by sorting the dataset on different sets of candidate columns, and running the query workload on the resulting sorted data for evaluation. However, in practice for large datasets, a single sort of the data could take hours. This is addressed in this work by introducing a *Clustering Quality Evaluator*. The Evaluator sorts a sample from the database on a candidate set of columns (which is much faster compared to sorting the entire dataset), and can then extrapolate performance on the sample to performance on the entire dataset. However, running the entire workload on a sample could be expensive as well, particularly when there are a large number of candidate solutions to search through. To address this, this work introduces a *Query Workload Modeler*, that builds a model of the queries. By substituting the entire dataset for a sample, and the workload queries for a query workload model, alternative clustering solutions can be compared in seconds instead of hours.

More formally, given a workload, this paper presents a methodology for finding a set of columns from the Fact table for linear clustering, combined with a set of columns to create zone maps on. The key components of this methodology are:

(1) *Query Workload Modeler*: The Query Workload Modeler analyzes the queries, and builds a model describing them. Instead of running the actual queries in the workload, which is much more expensive, this model is then used to evaluate the performance of queries for different clustering solutions.
(2) *Clustering Quality Evaluator*: The Clustering Quality Evaluator constructs a sample for the dataset. It then uses this sample to evaluate the performance of any candidate clustering solution. The key advantage of taking a sample is that it can be sorted quickly along different columns. So different clustering solutions can be evaluated without significant use of time or resources. The Evaluator works in conjunction with the Query Workload Modeler (1), which substitutes queries with a much cheaper query model.
(3) *Solution Search Algorithm*: A search algorithm then uses the Clustering Quality Evaluator mentioned above to search through the space of clustering solutions to find a good solution (the phrase clustering solution here refers to a set of columns to perform linear/interleaved clustering on).
(4) *Zone maps Recommender*: The Zone Maps recommender identifies any additional columns to build zone maps for (apart from the columns chosen for clustering).

For simplicity, in this document we assume that the database schema has a single Fact table we want to cluster (as is often the case for Star schema datasets). For workloads with multiple Fact tables (such as Snowflake schema datasets), this procedure is executed independently for each Fact table.

5 APPROACH DETAILS

The method requires the following information as input:

(1) A populated database instance consisting of Fact and Dimension tables, and a set of representative queries for the database.
(2) A list of Fact tables to cluster and build zone maps for. For each of these Fact tables, a list of Dimension tables that join

with the table, and a set of Join conditions J, describing which column of the Fact table joins with which column of each Dimension table.

While the algorithm requires a list of Fact and Dimension tables, this does not necessarily require human intervention. For example the Oracle RDBMS is capable of automatically identifying Fact and Dimension tables in the workload. Given this input, the auto-clustering recommendation procedure proceeds through the following steps:

(1) *Query Workload Modeling*: This step, consisting of the following two sub-steps, is first executed by the Query Workload Modeler.
   (a) *Column Pruning*: Analyze the queries to identify columns that are frequently used in queries. Use this information to heuristically prune bad candidate columns, and identify potentially good ones for clustering. This step also assigns a weight to each database column. The weight is a heuristic measure of the importance of the column to the workload.
   (b) *Workload Model Creation*: Use information from the first step is used to create a probabilistic model of the workload. This model includes information about column importance, as well as filter predicate characteristics for each column (specifically, selectivity and heterogeneity).
(2) *Sample Creation*: Take a sample of the candidate columns from the Fact and Dimension tables. This step requires joining a random sample from the Fact table with the entirety of the Dimension tables.
(3) *Clustering Quality Evaluator Initialization:* The Clustering Quality Evaluator takes as input a data sample that has undergone a linear or interleaved clustering on certain columns, and gives a numerical estimate of the effectiveness of the clustering. The Evaluator is used by the next step.
(4) *Solution Search Algorithm:* This step uses a greedy algorithm to search through the space of possible data clusterings, to find a good clustering solution. The Clustering Quality Evaluator created in Step 3 is used to evaluate and compare different clusterings.
(5) *Zone Maps Recommendation*: The recommendation step identifies additional columns (apart from the columns selected for clustering) to create zone maps on, given the chosen clustering solution.

The next five sections describes the above steps in order.

6 QUERY WORKLOAD MODELING

This process consists of two steps: an initial pruning step to reduce the amount of work done, followed by creation of the workload model from the pruned data.

6.1 Column Pruning

This is a pruning step, where columns that are not used as filter columns in a significant portion of queries are removed. At present, all columns that do not feature in at least 10% of the queries in the workload are removed from further consideration. The 10% level was selected as a heuristic, based on initial experiments to identify a reasonably selective cutoff, without removing most columns from consideration. For the columns that are not pruned out, we calculate a weight representing the importance of the column. The weight assigned to a table column is the number of queries in the workload, for which there is a filter clause on this column. This value is normalized across all columns, so that it sums to 1.

6.2 Workload Model Creation

At the end of the previous step, we have a list of candidate columns, along with a weight assigned to each of them, representing their importance. The weights are normalized to add to 1, so that the weights can be treated as a probability distribution.

Besides column importance, we consider another piece of information about each column with at least one equality filter predicate, which we call *heterogeneity*. Heterogeneity for a column is defined as the frequency distribution of the number of filtering terms observed for equality predicates on the column (for example, the query predicate "Where State = 'CA' or State = 'TX' or State = 'WA'" has three filtering terms), or the in-list size if the predicate is an in-list. Note that heterogeneity is the distribution of the number of filtering terms in the *query predicates* on the column. It is thus a characteristic of the queries, and not the data. Usually this value can be extracted from the query text, or the 'Predicate Information' section of the query plan. Formally, heterogeneity for a column c is represented as a set of 2-tuples $h_c = \{d_c^i, f_c^i\}$, where f_c^i is the normalized frequency with which d_c^i number of filtering terms are observed in predicates on c. As an example, if our workload has two queries on column State, where one of them has the predicate described above with 3 filtering terms, and the other query has a predicate "Where State = 'CA'" (with 1 filtering term), this would be represented by the heterogeneity model for the column as $\{(3, 0.5), (1, 0.5)\}$.

Heterogeneity is important because clustering can be ineffective against filter predicates with a high average heterogeneity. This is because, if the filter predicate accepts many distinct filtering terms, a large number of zones may need to be visited to find all relevant rows, irrespective of clustering or zone maps (as each zone is likely to have at least one of the filtering values).

Combining the column weights and heterogeneity together creates our workload model. A query workload can be represented as a set of 3-tuples: $Q = \{(m_c, w_c, h_c)\}$, $c = 1 \ldots n_C$, where n_C is the number of columns of interest, m_c is the name of the c^{th} column, w_c is its weight, and h_c is the heterogeneity of the filter predicates on the column (which is a set as described above).

The workload model is a simplified probabilistic model of how queries are generated for the dataset. At each step, a database column is selected, where the probability that column c is selected is w_c. Following this, a filter predicate is created for the selected column with a certain number of filtering terms (each query is assumed to have exactly one filter predicate). The number of filtering terms follows the distribution h_c, so that the probability that d_c^i filtering terms are generated is f_c^i. All distinct values in the column are independently drawn, and for each draw, all are equally likely to be selected as the filtering term.

Heterogeneity is however not a useful concept for range queries because, as mentioned above, we model equality predicates as generated by randomly selecting individual query filtering term values, where each value is drawn independently of the others (for example,

a predicate with 3 filtering terms for the State column is modeled as three iid draws from the possible set of 50 values). This assumption breaks for range queries as the filtering terms are often consecutive (and so not iid). A different model is used for range queries, which is discussed in Section 8.1.1.

7 SAMPLE CREATION

Once the candidate columns have been identified, we create a materialized sample of the columns. This sample is used to evaluate alternate clustering options, and later to identify the zone map columns. Given a set of candidate columns $C = \{c_1, \ldots, c_i \ldots, c_{n_C}\}$, a Fact table F, a set of Dimension tables D, and a set of Fact-Dimension join conditions J, we use the following query to create the sample:

```
Create table clustering_sample as
Select rowid row_id,C
From F Sample($sampling_percentage),D
Where J;
```

The join conditions J are left outer joins, instead of equi-joins, that is often the default for Fact-Dimension joins. The reason for this is that we do not want to exclude a Fact table row from the sample just because it does not join with one Dimension table, if it joins with other tables. In experiments, generally a sampling percentage of 0.1% to 0.2% was sufficient. The goal is to have at least 20 rows per zone in the sample, as it was empirically determined that having approximate these many rows per zone was sufficient to get reasonable estimates of the zone's min and max values. Also note that we create a random sample of rows. A popular alternative sampling option provided by many databases is a block sample, which is usually faster. However, we use chose not to use a block sample as within-block correlations between rows could impact the solution quality.

The reason we save row-ids as part of the samples, is that clustering by row-id allows us to analyze the default data organization of the original Fact table, in the absence of any other clustering. That is, before we try to identify clustering columns, we cluster the sample by row-id to set a baseline for performance: the case where no clustering is applied at all. This is done because applying any clustering solution to a table is expensive, and we want to cluster the entire table only if the predicated performance is at least a certain percentage better than the baseline (the default setting for this improvement is 10%).

8 CLUSTERING QUALITY EVALUATOR MODEL

The Clustering Quality Evaluator uses the sample taken from the dataset, and combines it with a probabilistic model that uses the sample to evaluate the quality of a given clustering solution. The probabilistic model is described in this section.

We estimate the quality of a clustering solution as the expected (average) number of zones visited for a randomly chosen query from the workload. To calculate the expected number of zones visited for a query/predicate (since the query generation model assumes exactly one predicate being generated for each query, we use query and predicate interchangeably in this Section, based on readability), consider the model introduced at the end of Section 6.2. For each candidate column for clustering/zone-map, we have a set of 3-tuples: $Q = \{(m_c, w_c, h_c)\}$, $c = 1 \ldots n_C$, where n_C is the number of columns, m_i is the name of the c^{th} column, w_c is its weight, and h_c is the heterogeneity set of the filter predicates on the column.

We first start with a simpler problem: estimating clustering quality given the entire table (instead of a sample). The result can then be extended to estimate clustering quality given a sample, after making some simplifying assumptions. So, instead of a sample, consider a Fact table (not a sample) that has already been clustered on a set of columns, and zone map have been constructed on some set of columns as well. How do we estimate the the expected number of zones accessed by a randomly chosen query from workload Q?

8.1 Clustering Quality Given Entire Table

We formalize the problem as follows: let A_Q be a random variable representing the number of accesses to a table if a workload Q is executed. We want to calculate $E[A_Q]$, the expected number of accesses given one randomly generated query from Q. As mentioned in Section 6.2, the workload Q is modeled as a probability distribution over columns, where the probability of a query on a column is proportional to its weight w_c. So we can calculate $E[A_Q]$ as:

$$E[A_Q] = \sum_{c \in Q} w_c E[A_c] \tag{1}$$

Here $E[A_c]$ is the expected number of zone accesses for a query on column c. Next, we address how to calculate $E[A_c]$ for each column. This requires an analysis of how the column's filter predicates' heterogeneity impacts the number of zones accessed.

Let Z be the set of zones on the Fact table. For now, assume that a zone map exists for all candidate columns. For a zone $z_j \in Z$, let $\mathbf{1}_{jc}$ be an indicator function representing the event that a random query referencing column c (from the workload) would access zone z_j. We want to calculate $E[A_c] = E\left[\sum_{z_j \in Z} \mathbf{1}_{jc}\right]$.By linearity of expectation, we know $E\left[\sum_{z_j \in Z} \mathbf{1}_{jc}\right] = \sum_{z_j \in Z} E\left[\mathbf{1}_{jc}\right]$. So, we can write:

$$E[A_c] = \sum_{z_j \in Z} E\left[\mathbf{1}_{jc}\right] \tag{2}$$

In words, summing the expected number of accesses to a single zone across all zones, gives us the expected number of zones visited. Note that $E\left[\mathbf{1}_{jc}\right]$, the expected number of visits to a single zone for a query, is always less than 1, so this is essentially the probability that the zone is accessed.

Now consider a single zone $z_j \in Z$, for which minimum and maximum values for column c are known (which would be the case, if a zone map was created for the column). We write the minimum and maximum value for a zone z_c^j as $\min(z_c^j)$ and $\max(z_c^j)$ respectively. To calculate $E\left[\mathbf{1}_{jc}\right]$ for the zone, we define a function *NDV distance* $V^c(x, y)$, which given two distinct values x and y from a column c, returns the *number of distinct values* (or NDVs) that lie inclusively between x and y in sorted order.

So, for example, for a column S that contains all 50 states, the function $V^c(\text{'}CA\text{'}, \text{'}FL\text{'}) = 5$, as it includes the following set of states: {'CA','CO','CT','DE','FL'}. Note that this function returns only the number of distinct values between two given values, and is not

impacted by the frequency with which each value occurs. We also introduce another function at this point: the *sample NDV distance* function $V_s^c(x, y)$ returns the number of distinct values between two keys in a sample. So while the NDV distance function provides the ground truth distance value, the sample NDV distance function $V_s^c(x, y)$ provides an estimate based on a sample. The $V_s^c(x, y)$ function is used in Section 8.2.

Then $E[\mathbf{1}_{jc}] = P[\mathbf{1}_{jc} = 1]$ is calculated as:

$$E[\mathbf{1}_{jc}] = \sum_{(d_c^i, f_c^i) \in h_c} f_c^i \left[1 - \left(1 - \frac{V^c(\min(z_c^j), \max(z_c^j))}{\text{NDV}_c} \right)^{d_c^i} \right] \quad (3)$$

In the above equation, NDV_c represents the number of distinct values in column c in the dataset. The summation on the right-hand-side is over all possible values of d_c^i (number of distinct filtering terms in a predicate), weighed by their respective probability (f_c^i). The probability that a query will access a zone z_j is 1 minus the probability that none of the d_c^i keys in the query appear in the zone. The probability that a randomly chosen key does not appear in a zone is given by $1 - \frac{V^c(\min(z_c^j), \max(z_c^j))}{\text{NDV}_c}$. Assuming the keys are chosen randomly, and all keys are equally likely to be chosen, this value is raised to the power of d_c^i to give the probability that none of randomly chosen d_c^i keys appear in the zone.

Equation 3 represents the expected number of accesses to a single zone for one randomly generated query on a randomly chosen column. So, to trace back our process, to calculate the total expected number of zones accessed for a random query, we first use eq. 3 to calculate the expected number of accesses to each zone, for each column. We sum this value across all zones using eq. 2 to calculate $E[A_c]$, the total expected number of zones accessed for each column. We then do a weighted sum across all columns using eq. 1, to calculate the expected number of zones accessed, for a randomly chosen column from the workload.

8.1.1 Zone Accesses for Range Predicates. The heterogeneity-based model works well for equality predicates, but it is less suitable for range predicates. While the model can be trivially extended to include range predicates (a range predicate with k distinct values can be modeled as a filter predicate with heterogeneity k), in practice this is likely to vastly overestimate the number of zone accesses required if the table is clustered on the range predicate column. This is because unlike our model assumption, the filtering terms in a range query are often consecutive and not independently selected. However, an alternate model may be used to estimate zone accesses for range queries, which is described below. While this model is currently not part of the existing implementation, it is planned as a future enhancement.

Given a set of range predicates on a single column in a workload, we can calculate the *mean range size* r_μ of predicates on the column. The mean range size is calculated as the sum of the number of distinct values observed for each range predicate on the column in the workload, divided by the total number of range predicates on the column. Treating this calculated value r_μ as representative of range predicates on the column, the probability that a predicate of range size r_μ would access a zone z_j is given by:

$$P[\mathbf{1}_{jc} = 1] = \min \left(\frac{r_\mu + V^c(\min(z_c^j), \max(z_c^j))}{NDV_C}, 1 \right) \quad (4)$$

The reasoning behind this equation is as follows: consider a range predicate of the form 'Where $c \geq r_{min}$ and $c \leq r_{max}$', where the range size is r_μ. A zone will need to be accessed for this range predicate if r_{min} is either: a) one of the values within the minimum and maximum value of the zone, or b) at most r_μ less than the minimum value in the zone. The number of possible values that r_{min} can take that lie within the zone is given by $V^c(\min(z_c^j), \max(z_c^j))$. The number of possible values less than $\min(z_c^j)$ that r_{min} can take, that would still require a zone access, is given by r_μ. Adding these together and normalizing gives us the probability of accessing a zone given a range query of size r_μ. The min function is added to ensure that the probability value is within zero and 1.

For columns with both equality and range predicates, the probability of accessing a zone can be calculated as a weighted mean of the probability of access for equality (eq. 3) and range predicates (eq. 4 above), where weights are based on the fraction of queries on the column with an equality predicate, versus the fraction of queries with a range predicate.

As mentioned above, inclusion of the range query model is currently planned as a future enhancement. Instead, to avoid overestimating the expected number of zone accesses required, the heterogeneity of all range queries is currently set to 1.

8.2 Extension to Clustering Quality Given a Sample

To estimate $E[A_Q]$ based on a sample, we scale down the size of each zone based on the sampling probability. For an original table zone size of S_Z rows, and a sampling probability p (so that $\frac{p}{100}$ % of the data is sampled), we set the zone size on the sample to $p * S_Z$. Scaling down the zone size in this manner gives us a random sample of keys that belong to the zone. Our goal is to estimate the following statistic from eq. 3, as we have values for the rest of the terms in eq. 3. We write this statistic as R_c.

$$R_c = \frac{V^c\left(\min\left(z_c^j\right), \max\left(z_c^j\right)\right)}{\text{NDV}_c} \quad (5)$$

The numerator of this term is the number of distinct values in a zone, while the denominator is the number of distinct values in the entire column. Estimating the number of distinct values in a population from a sample is a well-known problem in statistics, with an algorithmic solution provided by Charikar *et al* [2]. However, we take an alternate approach to estimating R_c, based on some simplifying assumptions, due to the following reasons: a) the algorithm provided in [2] is an iterative algorithm, and executing it for each zone can be expensive, and b) probably due to the small sample size available per zone, the algorithm [2] accuracy compared to our alternate approach was not enough to impact the clustering solutions found.

Our approach to estimate the number of distinct values is as follows: we start with an estimate of the endpoints ($\min(z_c^j)$ and

$\max(z_c^j)$) of each zone (though this estimate is likely to be an overestimate for the minimum, and an underestimate for the maximum). Using these endpoints, we estimate R_c as follows:

$$\hat{R_c} = \frac{V_s^c\left(\hat{\min}\left(z_c^j\right), \hat{\max}\left(z_c^j\right)\right)}{\hat{\text{NDV}}_c} \tag{6}$$

The values $\hat{\min}(z_c^j)$ and $\hat{\max}(z_c^j)$ represent the estimates of the minimum and maximum for a zone extracted from the sample (essentially the minimum and maximum value observed for the zone in the sample), while $\hat{\text{NDV}}_c$ represents the number of distinct values observed in the sample. Also, eq. 6 uses the *sample* NDV distance function V_s^c. With some simplifying assumptions, we show that $\hat{R_c}$ can be considered a reasonable approximation for R_c.This is addressed next.

Let's say we only see $\hat{NDV_c} = k \cdot NDV_c$ distinct keys in our sample, where k is a fraction. We make the following simplifying assumption: the fraction k is constant across zones. Then, $\hat{R_c}$ can be used as an approximation for R_c, because:

$$\hat{R_c} = \frac{k \cdot V^c\left(\min\left(z_c^j\right), \max\left(z_c^j\right)\right)}{k \cdot \text{NDV}_c} = R_c \tag{7}$$

The above assumption means that the keys missing from the sample are distributed across zones in proportion to the zone's sample NDV. This seems to be a reasonable assumption. Let's say we are comparing two zones for a column, and the first zone has only $2-3$ distinct keys, while the second zone has $20-30$ distinct keys. Then it seems reasonable to assume that the keys in the first zone have a higher frequency in the table, in which case it is likely that fewer keys were missed in the sample, compared to the second zone.

With these assumption, we can use equations 3, 2 and 1 over a sample to estimate the quality of a clustering. In our current experiments, we make a further simplification. Instead of representing the heterogeneity of a column's filter predicate as a distribution (as described in Section 6.2), we instead replace it with just the mean number of terms in the filter predicate, $\bar{d_c}$. This is a reasonable assumption if the filter heterogeneity is concentrated on a single value: for example, let's say almost all queries on a column have exactly one filter predicate, with few exceptions. With this simplifying assumption, eqs. 3, 2 and 1 can be combined into a single equation. While these simplifying assumptions must have some impact on the final result, our experimental results show that we are still able to get significant performance improvements over the baseline.

$$E[A_Q] = \sum_{c \in Q} w_c E[A_c] \tag{8}$$

We approximate $E[A_c]$ as:

$$\hat{E}[A_c] \approx \sum_{z_j \in Z}\left(1 - \left(1 - \frac{V_s^c(\hat{\min}(z_c^j), \hat{\max}(z_c^j))}{\hat{\text{NDV}}_c}\right)^{\bar{d_c}}\right) \tag{9}$$

Then, eq. 8 can be written as:

$$\Rightarrow \hat{E}[A_Q] \approx \sum_{c \in Q} w_c \cdot \left[\sum_{z_j \in Z}\left(1 - \left(1 - \frac{V_s^c(\hat{\min}(z_c^j), \hat{\max}(z_c^j))}{\hat{\text{NDV}}_c}\right)^{\bar{d_c}}\right)\right] \tag{10}$$

The key change between this equation and eq. 3 is that, instead of summing over all possible values of d_c, we only use the mean heterogeneity ($\bar{d_c}$).

9 SOLUTION SEARCH ALGORITHM

Algorithm 1 describes the algorithm used to find the best clustering solution. The algorithm is greedy: at each time-step, it selects the clustering with the lowest expected number of zone accesses, and expands this solution further, by adding a candidate column that is not part of the current solution, to it (the term candidate column refers to the set of columns that remain as possible additions to the clustering solution, after the column pruning phase is over). It assumes the existence of two functions:

(1) Sort_Linear_or_Interleave(S,new_cand_cols, sort_type_flag): The function has the capability to sort the sample, in linear or interleaved fashion, depending on the sort_type_flag. For interleaved clustering, it first creates a temporary virtual column by interleaving the columns in the list new_cand_cols, and then sorts the sample on these columns. This function is assumed to be provided by the RDBMS.
(2) Expected_Num_Accesses(Clustered_Sample), implemented within the Clustering Quality Evaluator (Section 8), that takes as input a clustered sample, and returns the expected number of zones visited for the workload, using eq. 10.

Apart from the input parameter, the function Expected Num Accesses (Clustered_Sample) also has the following information related to the workload available to it:

(1) The weight w_c attached to each column.
(2) The mean number of distinct filtering terms ($\bar{d_c}$) on the column.

After we have finished running Algorithm 1, we have our solution of columns to cluster on, as well as the expected number of zone maps visited for a randomly chosen query, given this clustering. The time complexity of the algorithm is of the order $O(n \log n)$, where n represents the number of rows in the sample. For linear clustering solutions search, the dominant cost is certainly the cost of sorting the sample. For interleaved clustering, the cost of interleaving the columns is another substantial cost to consider. In practice though, since the algorithm runs on a sample with a simplified query model, the cost of execution is not significant.

10 ZONE MAP COLUMN IDENTIFICATION

At the end of Section 9, we have a clustering solution, along with an estimate of the clustering quality $E[A_Q]$ (eq 8). However, this solution assumes that zone maps exist on all candidate columns. While it is theoretically possible to maintain zone maps for all columns, it is not possible in practice due to the increased maintenance overhead. For this reason, in this step, we prune this list of candidate zone map columns, removing columns that do not have a significant impact on $E[A_Q]$.

Algorithm 1: Greedy algorithm to identify clustering columns.

input : S: A sample with columns $C = \{c_1, \ldots, N_C\}$
τ: Maximum number of clustering columns allowed (default = 4)
Δ: Minimum percentage improvement required for column inclusion (default = 5%)
sort_type_flag: Linear/Interleaved

output : A 2-tuple *soln*, where soln._1 is the list of clustering columns and soln._2 is the clustering quality.

Initialization:

Set candidate solution queue $U \leftarrow \phi$
Set S_clus ← Sort_Linear_Or_Interleave(S,rowid,sort_type_flag)
Set baseline_soln ← Expected_Num_Accesses(S_clus)
Add 2-tuple (ϕ,baseline_soln) to U

Loop:

do
  Set cur_soln ← $u \in U$ with smallest value of u._2
  Set cur_quality ← cur_soln._2
  Set cur_clus_cols ← cur_soln._1
  Set new_soln_found ← False
  for $c \in C - cur\_clus\_cols$ **do**
    Set new_cand_cols ← cur_clus_cols.append(c)
    Set S_clus ← Sort_Linear_Or_Interleave(S,new_cand_cols,sort_type_flag)
    Set new_clus_quality ← Expected_Num_Accesses(S_clus)
    Set impr ← (cur_quality−new_clus_quality)/(cur_quality)
    if $impr > \Delta$ **then**
      Add (new_cand_cols,new_clus_quality) to U
      Set new_soln_found ← True
while *len*(*cur_soln.cols*) $< \tau$ *and new_soln_found*

Result:

Set soln ← $u \in U$ with smallest value of u._2
return *soln*

If a zone map exists on column c, an estimate of the expected number of zone accesses $E[A_c]$ for a randomly generated predicate on c is given by eq. 9. If we do not have a zone map on a column, all zones will be accessed for a predicate on the column, so $E[A_c] = N_Z$, the number of zones in the table. This is the baseline we need to compare against. Note that the function Expected_Num_Accesses estimates the expected number of zone accesses $E[A_c]$ for each column c as part of its internal calculations. It is straightforward to modify this function so that it can individually return its estimate of the number of accesses for each column.

Assuming we modify Expected_Num_Accesses to return a vector $\vec{A_C}$, where the c^{th} element of this vector is $\hat{E}[A_c]$, the expected number of zones accessed for a randomly generated predicate only on column c. Note that $\vec{A_C}$ is the input to eq. 8. Then the impact of not creating a zone map on column c can be calculated as follows: replace $\hat{E}[A_c]$ with N_Z in $\vec{A_C}$, and recalculate $E[A_Q]$ using eq. 8. We write this recalculated value as $E[A_Q|\neg c]$. Then we calculate the impact of the zone map on column c as:

$$\text{Impact}_c = \frac{E[A_Q|\neg c] - E[A_Q]}{E[A_Q]} * 100 \tag{11}$$

Impact_c is the percentage increase in total zone map accesses as per our model, if a zone map is not created on c. We only create zone maps on columns where the impact is above a certain threshold. Currently the impact is set to 5%, but this value can be changed by the user. Note that while the threshold for clustering on a column is 10%, the threshold for zone map creation is lower, as clustering is a more expensive operation than zone map creation.

At present, however, the Oracle database does not support the identification of additional zone map columns: zone maps are built only on columns selected for clustering. One reason for this is that existing Exadata [12] access structures such as storage indexes [13] already have functionality to identify important columns to maintain per column minimum/maximum information on (though this feature is not available on non-Exadata machines), due to which zone map column identification might be somewhat redundant functionality. However, in Section 12, we do some additional analysis outside the database system, to measure the impact of creating additional zone map columns using the approach described in this section.

11 DATABASE VERIFICATION

This section describes how the RDBMS uses the recommendation provided by the auto-clustering algorithm. Since at present, it is not possible for the RDBMS to identify apriori (that is, before taking a sample) whether linear or interleaved clustering would be the appropriate choice for a workload, the RDBMS uses a common sample to arrive at both linear and interleaved clustering solutions. At this point, the RDBMS needs to decide whether to use the linear or interleaved clustering solution. The Solution Search algorithm does provide a clustering quality estimate as part of its output (see Figure 1), and theoretically this value can be compared across solutions. However, the RDBMS does not rely on this value for its decision, as clustering is an expensive process that is difficult to re-do or reverse.

Instead, the RDBMS creates a clone of the original workload (all tables and auxiliary strucures) to verify the solutions provided. It implements both the linear and clustering solutions on this clone and measures their performance across multiple metrics. A clustering solution is accepted only if it shows at least a 10% improvement on the number of logical I/Os performed, and the number of queries that regress from the baseline is below a certain threshold. If both clustering solutions are acceptable with these conditions, then the one that does better on the logical I/O metric is accepted. The verification process runs in the background when sufficient idle resources are available. It has the capability to stop and resume multiple times. While this process is expensive, it guarantees that the clustering performed by the system would be acceptable to the user. In addition to being able to use a background process for clustering, the RDBMS also provides a manual interface that can invoke both recommendation and verification process. This has

Dataset Name	Table Name	Dataset Size	Number of Tables	Number of Queries
Workload G	Snowflake	100-500GB	50	500-1000
Workload H	Star	50-200GB	20	50-150

Table 1: Customer Datasets Description

the advantage that the user can allocate resources based on his/her discretion, and may not require suspension and resume.

It may be possible to reduce the cost of the verification process, for example by running the workload on a sample, and extrapolating the clustering performance from the results. This is a problem we intend to investigate in future work.

12 EXPERIMENTAL RESULTS

We present results of experiments run on the TPCH public benchmark [16] , and two internal benchmark customer workloads (referred to here as Dataset G and Dataset H). As a baseline for comparison, we use the performance of the RDBMS on these workloads when the data is unclustered (the default layout), and in the absence of zone maps. We believe this baseline allows us to measure exactly the impact of auto-clustering, in combination with zone maps, compared to not having these structures for a workload.

12.1 TPCH (1 Terabyte) Benchmark

The auto-clustering algorithm was evaluated on a TPCH workload with a scale factor of 1000 (giving a dataset size of 1 Terabyte). For the experiment, one query was generated for each of the twenty-two TPCH query templates. The algorithm investigated both linear and interleaved clusterings for two of the largest tables in the dataset: LineItem and Orders. However, the interleaved clustering was rejected during validation, and is not presented here.

For the LineItem table, the algorithm suggested a linear clustering Linear($L\_Shipdate, L\_Quantity$). The clustering was accepted during the validation phase. The solution found by the algorithm seems intuitively reasonable, as L_Shipdate is a filter column in several of the twenty-two TPCH query templates. While there is no hierarchical relationship between L_Shipdate and L_Quantity, the L_Quantity column was added to the clustering because it provided incremental improvement for a subset of queries. A clustering solution was also identified by the algorithm for the Orders column, but was rejected in the validation phase. As described in Section 10, currently the database only builds zone maps on the clustering columns. However, additional analysis was performed outside the database code to identify further columns that might benefit from zone maps, using the approach described in Section 10. In the case of TPCH, the analysis suggested no further columns.

A standard Oracle Exadata machine was used for all performance experiments, and the degree of parallelism (DOP) was set to 16. The results are based on the average of three independent runs for each setting. Figure 1 shows the performance of the solution found by auto-clustering, to a baseline of non-clustered data. The three key metrics presented are: a) Elapsed Time, which is the total time the workload took to run, b) CPU Time, the time spent in the workload doing CPU operations, and c) the amount of logical I/O done while executing the queries. The results show a 24% reduction in the number of logical I/Os required, but an even higher approximately 40% reduction in Elapsed Time and CPU Time. A larger reduction in Elapsed Time and CPU Time is due to a more pronounced effect of zone map pruning resulting in better caching, memory utilization and data distribution traffic.

12.2 Customer Workloads

This section presents performance results on two customer datasets. For these datasets, we present results for both linear and interleaved clustering, as neither solution was rejected during verification for either dataset. Table 1 provides a brief description of these two datasets. The dataset descriptions are approximate due to legal constraints. While Workload H is a star schema dataset with one Fact table, Workload G is a snowflake schema dataset with multiple Fact tables. Figures 2 and 3 show the performance of linear and interleaved clustering in comparison to a baseline where the data is unclustered and no zone maps are constructed for these datasets.

Due to legal constraints, we cannot show the actual values for these statistics. The value on top of each bar shows the percentage improvement compared to the baseline. As can be seen from the figures, on both datasets, auto-clustering performs considerably better than the baseline on all three key metrics. Auto-Clustering directly impacts the amount of logical I/O, as the goal of clustering is to minimize I/O, and we see an $47 - 67\%$ improvement in this metric. However, as was the case with TPCH, it indirectly improves CPU and Elapsed Time as well, as it reduces the amount of data required to be processed by the CPU, and the overall execution time required.

For Workload H, the clustering solution found for both linear and interleaved clustering is identical and consists of clustering on two columns that have no obvious (hierarchical or otherwise) relationship to each other. The algorithm's estimate for the quality of the solution was also higher for interleaved clustering, compared to linear clustering. Based this, it is reasonable to assume that the auto-clustering algorithm was trying to balance different subsets of the workload, each of which was filtering on a different column. Unlike the TPCH dataset, where the solution was largely based on identifying a single important column, in this case the algorithm was able to identify that an interleaving of two competing columns was the best solution. As we did for for the TPCH workload, we used the approach described in Section 10 to identify any additional columns that might improve solution quality by more than our chosen threshold of 5%. But the analysis did not suggest any such additional columns.

While Workload G has multiple Fact tables, clustering was recommended for only one table. For this workload, the clustering solutions found by linear and interleaved clustering differ by one column. While linear clustering clusters on two unrelated columns (X and Y), the interleaved clustering recommendation was for three

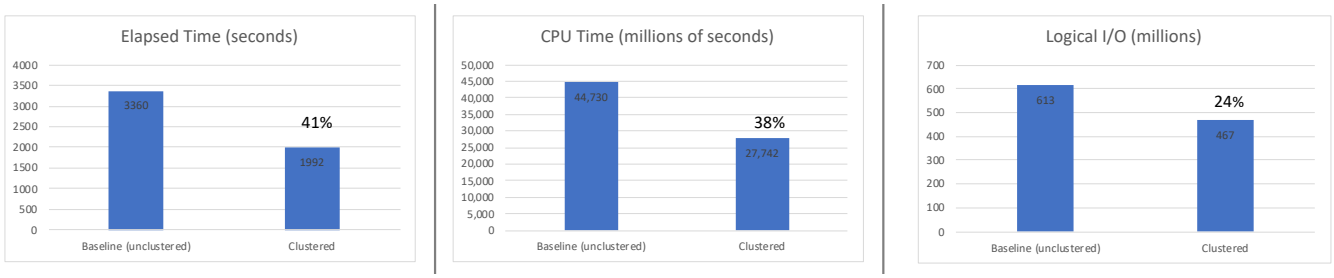

Figure 1: Auto-clustering (linear) performance, compared to baseline for TPCH benchmark (1TB)

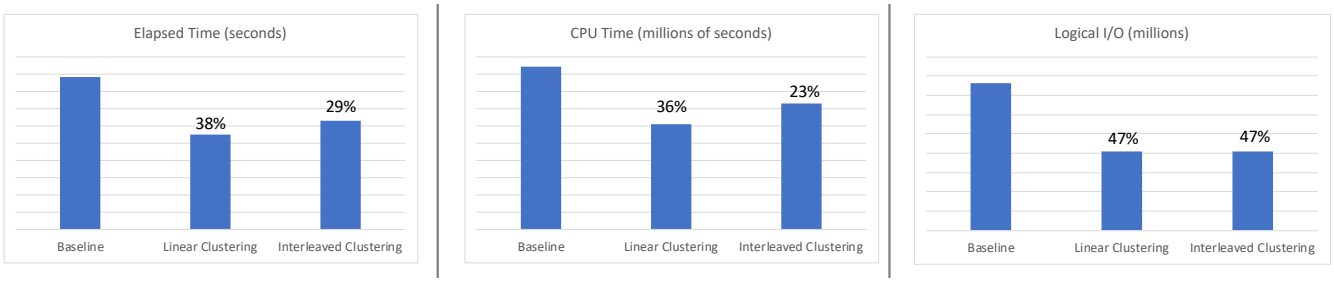

Figure 2: Linear and Interleaved auto-clustering performance, compared to baseline for customer workload G

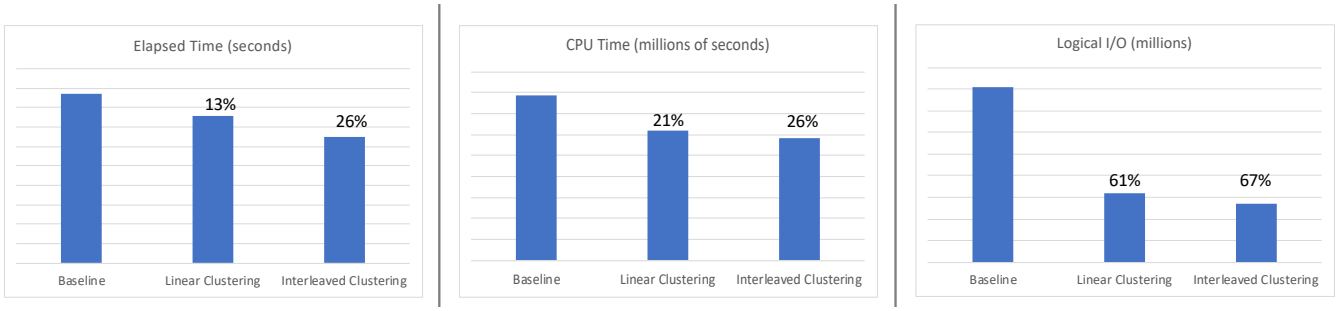

Figure 3: Linear and Interleaved auto-clustering performance, compared to baseline for customer workload H

columns (X, Y and Z). For linear clustering, addition analysis (Section 10) suggested one additional column K as a candidate for zone map construction, with an overall estimated clustering quality improvement of slightly more than 5%. Experiments done with this additional zone map column showed a 3% improvement in CPU time and DB time, and a 2% improvement in logical I/O, compared to the standard auto-clustering solution with zone maps only on clustering columns (the plots show results for the standard auto-clustering solutions, as that is the solution currently implemented). For interleaved clustering, no further columns were suggested for additional zone maps.

Interestingly, in terms of performance, linear clustering outperforms interleaved clustering on Elapsed Time and CPU Time, while in terms of Logical I/O, both approaches show similar behavior. We believe the underperformance of interleaved clustering in this case might be due to the higher cost of uncompressing the tables in the case of interleaved clustering. This issue underscores the importance of database verification (Section 11), as there are many factors that can impact the performance of a workload, and it is not possible to take all of them into account in a mathematical model. Note, however, that in this case as well, both linear and interleaved clustering significantly outperform the baseline.

13 CONCLUSIONS AND FUTURE WORK

Zone maps are data structures commonly found in data warehousing solutions, and are most effective when combined with a suitable data clustering. However to the best of our knowledge, there is no automated methodology to identify a good clustering for a workload. This goal of this paper is to fill this gap by presenting a new approach for automatically identifying a suitable clustering given a workload. Our results on the TPCH benchmark and two customer workloads show that automated clustering (auto-clustering) can provide significant performance gains. This underscores the importance of zone maps as a data access structure, as well as the need for a good clustering solution to support them.

In future work, we plan to remove some of the assumptions and approximations we had to make to improve performance during the solution search phase. For example, a more sophisticated model of the query workload (for example, using a more elaborate approach to model heterogeneity) has the potential to improve results. We also plan to explore more sophisticated search algorithms in order to find the best clustering solution, as opposed to a straightforward greedy algorithm. A solution search algorithm that considers both immediate and long-term consequences of choosing a column for clustering could potentially find much better clustering solutions. Another avenue we plan to explore are hybrid clustering solutions, such as combining multiple linear clustering solutions by interleaving them.

REFERENCES

[1] Nigel Bayliss. 2014. *Optimizing Table Scans with Zone Maps.* https://blogs.oracle.com/datawarehousing/optimizing-table-scans-with-zone-maps [Online].

[2] Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 2000. Towards Estimation Error Guarantees for Distinct Values. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Dallas, Texas, USA) *(PODS '00)*. Association for Computing Machinery, New York, NY, USA, 268–279. https://doi.org/10.1145/335168.335230

[3] Jef Claes. [n. d.]. *Amazon Redshift - Fundamentals.* https://s3-eu-west-1.amazonaws.com/cdn.jefclaes.be/amazon-redshift-fundamentals/aws-redshift-fundamentals.html [Online].

[4] Jialin Ding, Umar Farooq Minhas, Badrish Chandramouli, Chi Wang, Yinan Li, Ying Li, Donald Kossmann, Johannes Gehrke, and Tim Kraska. 2021. Instance-Optimized Data Layouts for Cloud Analytics Workloads. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 418–431. https://doi.org/10.1145/3448016.3457270

[5] IBM Documentation. 2016. *IBM PureData System for Analytics.* https://www.ibm.com/support/knowledgecenter/SSULQD_7.2.1/com.ibm.nz.adm.doc/c_sysadm_zone_maps.html [Online].

[6] Oracle Documentation. 2021. *Oracle Data Warehousing Guide: Attribute Clustering.* https://docs.oracle.com/en/database/oracle/oracle-database/21/dwhsg/attribute-clustering.html#GUID-7B007A3C-53C2-4437-9E71-9ECECF8B4FAB [Online].

[7] Snowflake Database Public Documentation. 2006. *Automatic Clustering.* https://docs.snowflake.com/en/user-guide/tables-auto-reclustering.html [Online].

[8] Snowflake Database Public Documentation. 2023. *Clustering Information Maintained for Micro-partitions.* https://docs.snowflake.com/en/user-guide/tables-clustering-micropartitions.html#clustering-information-maintained-for-micro-partitions [Online].

[9] Snowflake Database Public Documentation. 2023. *Strategies for Selecting Clustering Keys.* https://docs.snowflake.com/en/user-guide/tables-clustering-keys.html#strategies-for-selecting-clustering-keys [Online].

[10] Stratos Idreos, Martin L Kersten, and Stefan Manegold. 2007. Database Cracking.. In *CIDR*, Vol. 7. 68–78. https://stratos.seas.harvard.edu/files/IKM_CIDR07.pdf

[11] G. M. Morton. 1966. *A Computer Oriented Geodetic Data Base; and a New Technique in File Sequencing.* Technical Report. IBM, Ottawa, Ontario, Canada.

[12] Oracle. 2024. *Oracle Exadata.* https://www.oracle.com/engineered-systems/exadata [Online].

[13] Oracle. 2024. *Storage Indexes on Oracle Exadata.* https://www.oracle.com/database/technologies/exadata/software/storageindexes [Online].

[14] Chapter 13 Oracle 12c Data Warehousing Guide. 2017. *About Zone Maps.* https://docs.oracle.com/database/121/DWHSG/zone_maps.htm#DWHSG8935 [Online].

[15] Zack Slayton. 2017. *Z-Order Indexing for Multifaceted Queries in Amazon DynamoDB.* https://aws.amazon.com/blogs/database/z-order-indexing-for-multifaceted-queries-in-amazon-dynamodb-part-1/.

[16] TPC. 2023. *The TPCH Homepage.* https://www.tpc.org/tpch/.

[17] Wikipedia. 2024. *Space Filling Curve.* https://en.wikipedia.org/wiki/Space-filling_curve [Online].

[18] Wikipedia. 2024. *Z-order Curve.* https://en.wikipedia.org/wiki/Z-order_curve [Online].

[19] Zongheng Yang, Badrish Chandramouli, Chi Wang, Johannes Gehrke, Yinan Li, Umar Farooq Minhas, Per-Åke Larson, Donald Kossmann, and Rajeev Acharya. 2020. Qd-Tree: Learning Data Layouts for Big Data Analytics. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 193–208. https://doi.org/10.1145/3318464.3389770

[20] Mohamed Ziauddin, Andrew Witkowski, You Jung Kim, Dmitry Potapov, Janaki Lahorani, and Murali Krishna. 2017. Dimensions Based Data Clustering and Zone Maps. *Proc. VLDB Endow.* 10, 12 (Aug. 2017), 1622–1633. https://doi.org/10.14778/3137765.3137769