SCRAP: A Statistical Approach for Creating a Database Query Workload Based on Performance Bottlenecks

James Skarie*, Biplob K. Debnath*, David J. Lilja*,and Mohamed F. Mokbel†
* Electrical and Computer Engineering Department, University of Minnesota, USA
† Computer Science Department, University of Minnesota, USA
{skar0059,debna004,lilja}@umn.edu
mokbel@cs.umn.edu

***Abstract*—With the tremendous growth in stored data, the role of database systems has become more significant than ever before. Standard query workloads, such as the TPC-C and TPC-H benchmark suites, are used to evaluate and tune the functionality and performance of database systems. Running and configuring benchmarks is a time consuming task. It requires substantial statistical expertise due to the enormous data size and large number of queries in the workload. *Subsetting* can be used to reduce the number of queries in a workload. An existing workload subsetting technique selected queries based on similarities of the ranks of the queries for low-level characteristics, such as cache miss rates, or based on the execution time required in different computer systems. However, many low-level characteristics are correlated, produce similar behaviors. Also, raw execution time as a metric is too diffuse to capture important performance bottlenecks. Our goal is to select a subset of queries that can reproduce the same bottlenecks in the system as the original workload. In this paper, we propose a statistical approach for creating a database query workload based on performance bottlenecks (SCRAP). Our methodology takes a query workload and a set of system configuration parameters as inputs, and selects a subset of the queries from the workload based on the similarity of performance bottlenecks. Experimental results using the TPC-H benchmark and the PostgreSQL database system, show that the reduced workload and the original workload produce similar performance bottlenecks, and the subset accurately estimates the total execution time.**

I. Introduction

With the rapid expansion of the internet, data is growing at an exponential rate. Businesses are building larger databases to cope with this enormous data growth rate [1]. The performance of a database is critical for the success of a business. The performance of a database system is highly dependent on the physical database design, some examples include, index selection, materialized views selection, and data partitioning [1]–[3]. The performance of a database also depends on the appropriate values of the configuration parameters.

Database administrators (DBAs) use query workloads to tune the physical design of databases to find the proper values of the configuration parameters. The TPC-C and TPC-H are the examples of workloads which are used to evaluate and tune a database system. One of the major problems with database benchmarking is the difficulty of setting database parameters and database layout which requires a complex skill set [4]. The difficulty of using benchmarks is the massive data size and the large number of queries in the workload. The TPC-H benchmark has 22 read-only queries and two update queries. The TPC-H benchmark database size varies from 1GB to 100TB. Many of the queries, for example, Q1, Q9, and Q22, require long time to execute. Also the execution time varies based on the database size and the existence of indexes. It is not unreasonable that, to reflect current growth of data, we have to increase the size of the TPC-H benchmark to 10EB (exa bytes).

To finish the evaluation and tuning within a feasible time, DBAs use *subsetting* to select a representative subset of the workload which reduces the execution time but preserves the fidelity of the original workload. The subset is selected based on the high level knowledge of the operational behaviors of the queries from the query execution plan, for example, either join bound or scan bound, or based on similarities between the ranking of different low-level metrics and execution time in different computer system. Two queries can produce similar high-level run-time behavior but can have different performance bottlenecks in the system. Moreover, many low-level metrics are not independent. Which is compounded with the problem of the execution time in different computer systems being a very coarse metric to capture performance bottlenecks of the system. Using a subset based on inappropriate metrics, can lead to a wrong conclusion of the systems performance bottlenecks. For example, subset workload may ignore a parameter which has a significant impact on the system performance. Misdirected tuning efforts increase total cost of ownership of a database system [5]–[8]. To address these problems, a sound methodology which can help DBAs to find an appropriate subset of the query workload to correctly identify the system performance bottlenecks parameters is needed.

Our goal in this work is to find a subset of the workload based on the performance bottlenecks of the constituent queries. To achieve this goal, we propose a statistical approach for creating a compact representational query workload from an original workload for a database system based on performance bottlenecks (SCRAP). In particular, SCRAP addresses the following problem: *Given a DBMS, a set of configuration parameters, a range of values for all parameters, and a query workload, find a subset of the workload which can produce similar performance bottlenecks as the original workload.* The Workload is modeled as a set of SQL statements, for example, SELECT, UPDATE, DELETE, INSERT, and CREATE VIEW statements. A workload can be a set of benchmark queries, for example, TPC queries, or can be a set of data manipulation language (DML) statements collected over a fixed amount of time using profiling tools.

SCRAP uses a *design of experiments* based PLACKETT & BURMAN (P&B) design methodology [9] to estimate the effect of a parameter on the database system performance. The main idea is to conduct a set of selected experiments based on a specification such that these experiments will provide a sampling of the entire search space. The parameter values are set only to the maximum and minimum values. Stimulating the system with extreme values for nondecreasing monotonic input will provoke the maximum response for each input. Another assumption we make here is that only single and two-factor parameters interactions are significant. Based on these assumptions, SCRAP employs P&B design which reduces the search space from exponential to linear. For each query, the effects of all parameters form a vector of effects. Based on the Euclidean distance among these vectors, SCRAP clusters similar queries in the same group. Finally, from each group, one query is selected to form the subset of the query workload.

SCRAP will help DBAs of all knowledge levels to identify performance bottlenecks in the system based on the statistical data. Using the TPC-H benchmark [10] and the PostgreSQL database system [11], we demonstrate that SCRAP selects a subset of the workload, which produces performance bottlenecks similar to the complete workload. Our contributions in this paper are:

- *A methodology for clustering database queries.* The queries of a database workload are clustered based on the similarities among their system performance bottlenecks parameters.
- *Providing experimental evidence.* Our experimental results show that the performance bottlenecks produced by the reduced workload are similar to the bottlenecks produced by the original workload.

The remainder of the paper is organized as follows: Section II describes the related work; Section III describes our experimental methodology and a brief overview of the statistical P&B design; Section IV describes SCRAP in detail; Section V describes the experimental setup; Section VI explains our results; finally, Section VII concludes the discussion.

II. RELATED WORK

Simplified database benchmarks have been previously used for analyzing the efficiency of the processor and memory configurations [4], [12]–[14]. The reduced benchmarks have been shown to produce similar results with a significant reduction in simulation time [14]–[16]. The existing work on reducing database workload can be classified into two categories: *scaling down benchmark size* and *subsetting*. SCRAP belongs to the second category.

Scaling down benchmark size: Dbmbench identifies that a benchmark can be reduced along three dimensions: database size, workload complexity, and concurrency level [13]. Simultaneously scaling along these three dimensions, Dbmbench produces μTPC-H and μTPC-C that more than 95% accurately captures the memory and processor performance of the online transaction processing and decision support systems workloads, respectively.

Subsetting: Studies in [14]–[16] use subsetting. They do not provide analysis of how the subsets are selected. Subsetting has been applied to the the TPC-H query workload [17], based on the Principal Component Analysis (PCA) of low-level performance metrics such as instruction count, cache miss rate. The drawback of this approach is that many low-level metrics are correlated and measure similar characteristics. Another group selected a subset of queries based on the similarity between the ranking of execution times of different computer systems for the queries of a workload [18]. Nonetheless, ranking based on the execution time in different computers systems is too diffuse to capture the performance bottlenecks. Moreover, this method cannot be applied for a single computer system. In contrast, SCRAP is a more fine-grained and clusters queries based on the similarity between the performance bottlenecks parameters for a single system.

SCRAP uses the P&B design methodology to estimate the effect of a configuration parameter on database system performance. Similar approach is used to select a subset of microarchitecture benchmarks [19]. Our novelties in this paper are the following: applying P&B method to a database system, as opposed to P&B applied in context of hardware, but we have applied it to software applications, showing how to cluster queries instead of clustering the whole benchmark programs, finding Euclidean distance using three different methods and showing all the methods produce almost similar results. Moreover, database queries have different behaviors from the microarchitecture benchmarks. Query signatures like TPC-H have input parameters where the input values are selected from a range of values. Depending on value selection, query execution plans differ drastically from each other. To find an accurate representation of the workload, DBAs have to consider multiple values for the query signature input parameters. This is very different from benchmark programs, where the benchmark programs behaviors are reasonably consistent.

III. DESIGN OF EXPERIMENTS BASED STRATEGY

SCRAP uses similarity between the effects of configuration parameters on the database system performance to cluster queries of a workload. To estimate the effect a parameter, SCRAP employs a *design of experiments* based approach. The main goal of *design of experiments* is to gather maximum information about the parameters of a system with minimum amount of time and effort [20]. A *full factorial design*, for example ANOVA, can be used to estimate the effects of all main parameters and all of their interactions. It requires an exponential number of experiments. In a database, there are hundreds of configuration parameters and each parameter can assume multiple values. For example, PostgreSQL has approximately 100 configuration parameters and every parameter can assume multiple possible values. If we assume that every parameter has only two possible values, in this case for applying a *full factorial design* methodology, we need to conduct 2^{100} experiments, which is not feasible in terms of time and effort.

In order to reduce number of experiments from exponential to linear, we make two assumptions. First assumption is that stimulating system with monotonic input parameter with extreme values will provoke maximum response for each input parameter. Second assumption is that only single factor and two-factor interactions are important to be considered. Based on these two assumptions, SCRAP employs a two-level factorial design named the P&B design with *foldover* [21], which accurately quantifies main effects and two-factor interactions using only a linear number of experiments.

To apply the P&B design with *foldover* to estimate the effects of N configuration parameters for a database query, we have to conduct $2X$ experiments, where X is the next multiple of four greater than N. Each experiment is conducted based on a specification given by the *P&B design matrix*, which has N columns and $2X$ rows. The entry [i,j] in the design matrix corresponds to the value to be set for the j-th parameter of the i-th experiment. Each entry in the matrix is either "+1" or "-1". "+1" means high value and "-1" means low value. The first row of the design matrix is selected from [9], based on the value of X. Next $X - 1$ rows are generated by right-cyclic shifting of the immediate preceding row. In the X-th row, each entry is set to "-1". The lower X rows are generated by reversing the sign of the entries of the upper X rows.

For each experiment, the execution time of a query is recorded. The effect of the j-th parameter is calculated by multiplying the [i,j] entry value with the query execution time for i-th experiments and summing all the effects across all $2X$ rows. The magnitude of the summed value of execution times gives an estimation of the effect of a parameter on the database performance for the corresponding query. The effects of all configuration parameters form a vector of effects for a query. The similarities between the effect vectors are used to cluster the queries of a database workload.

IV. Methodology

This section describes our methodology in detail. SCRAP consists of four phases. The first phase estimates the effects of all configuration parameters for the constituting queries of a workload, using the P&B design methodology. The second phase determines whether a query is sensitive to tuning or not, and forms a cluster consisting of all tuning insensitive queries. The third phase clusters the tuning sensitive queries based on similarities among performance bottlenecks. The fourth phase selects a query from each cluster to form a subset of the original query workload. The outline of the SCRAP methodology is given in Algorithm 1. Each phase is discussed in detail in the following subsections. At end of this section, we discuss how to select input parameters and query workload to improve the accuracy of the SCRAP.

Algorithm 1 The Outline of the SCRAP Algorithm for Finding a Subset of the Query Workload

```
1: Input: a query workload, a set of parameters, and their high and low values.
2: Output: a subset of the query workload which produces same performance bottlenecks as the input workload.
3: Estimate the effects of all parameters for each query of the workload.
4: Determine whether a query is sensitive to parameter tuning or not.
5: Cluster the sensitive queries.
6: Select a query from each cluster and insensitive query group to form the subset workload.
```

A. Effect Estimation

In the first phase of SCRAP, a design of experiments based P&B design with *foldover* methodology to estimate the effect of each parameter. Based on the number of input parameters, the *P&B design matrix* is formed. Each row of the design matrix corresponds to one experiment and column values in that row specify the values that need to be set for the parameters for the corresponding experiment.

For each query of the workload, we conduct experiments according to *P&B design matrix* specification and record the execution time of the query. The net effect a parameter is estimated by multiplying query execution time with the corresponding "+1" or "-1", summing across all rows, and finally taking the magnitude of the summation. The relative magnitude of the effect can be used to rank the parameters based on their impact on the database system performance.

B. Insensitive Query Identification

In a query workload, there are some queries for which the execution time varies very small amount with the change in parameters values. We define them as *tuning insensitive queries*. If the all execution times of a query are virtually equivalent for all the experiments conducted according to the *P&B design matrix* specification, it means that the corresponding query's performance is not affected by parameter value changes. All insensitive queries of a workload are clustered in a group. Coefficient of Variation (COV) of the execution times of a query is used to determine a query's sensitivity to parameter value change. The threshold of the COV should be selected based on the application expected behavior. Typically, a threshold of 0.05 for the COV is a good estimate.

C. Clustering Queries

In this phase, the tuning sensitive queries are clustered based on the similarities among the estimated effects of the parameters. This is done in three steps. First, the estimated P&B effects of the all configuration parameters for the sensitive queries are either normalized or ranked to from a vector of performance bottlenecks. Second, a similarity metric is defined. Third, a threshold of the similarity metric is used to cluster the queries.

1) Vector Formation: The effects of all parameters for a query is either normalized or ranked to form a vector of performance bottlenecks. These vectors are used to find similar queries. The vector can be formed in different ways. In this paper, we try the following three methods.

The **first** method is *the normalizing a effect with respect to the sum of effects*. For example, we have four parameters and a query has the P&B effects 142, 12, 182, and 145 for these parameters. The sum of the effects is 481. The normalized effects with respect to sum effects are 142/481, 12/481, 182/481, and 145/481, or 0.3, 0.03, 0.38, and 0.3, respectively. So the vector will be $< 0.3, 0.03, 0.38, 0.3 >$.

The **second** method is *the normalizing a effect with respect to the maximum effect*. For example, the maximum effect for the query in the earlier example is 182. Normalizing with respect to maximum effect gives us the effects of 142/182, 12/182,182/182, and 145/182, or 0.78, 0.07, 1.0, and 0.80, respectively. By normalizing the effect with respect to the maximum effect, we can estimate how each parameter affects performance compared to the maximum effect for a query. Here, the vector is $< 0.78, 0.07, 1.0, 0.80 >$.

The **third** method is to *rank* the queries according to relative magnitude the effects. A simple approach is to sort the effects descending order of their magnitude. However, if there are some effects which are almost similar, ranking method inaccurately assigns different ranks to them. Often several of the parameters are close enough in sensitivity that the accuracy of P & B limits the ability to select which rank is greater, the queries with similar sensitives are grouped to remove this difficulty. To overcome this problem, we normalize the sensitivity values with respect to the maximum effect, round the normalized effect up to the first decimal point, and assign all parameters with the same normalized effects the same rank. The choice of rounding up to the first decimal digit is selected to ensure the queries that are less then one order of magnitude of the maximum query are assigned to ranking group. The ranking assigned in a group is the largest ranking of a parameter in that group. For the example query normalized with respect to the maximum effect gives 0.8, 0.1, 1.0, and 0.8 and and the corresponding vector is $< 3, 4, 1, 3 >$.

2) Metric Selection: SCRAP uses the Euclidean distance between two queries to determine clustering. The Euclidean distance between the normalized P&B effects vectors (or rank vectors) for each query is used for finding the similarities between queries. The smaller the Euclidean distance is, the more similar the corresponding queries are. If there is N parameters and $X = [x_1, x_2, \ldots, x_N]$, and $Y = [y_1, y_2, \ldots, y_N]$ represents the effect or the ranked vector for query Q_X and Q_Y, respectively. The Euclidean distance between them is defined as: $\left[(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_N - y_N)^2\right]^{1/2}$.

3) Cluster Formation: A dendrogram is used to graphically illustrate clusters produced by a clustering algorithm. The Euclidean distance is calculated between each query, and placed into a distance matrix. The groups with the smallest Euclidean distances are grouped first, and the grouping continues until there is only one cluster left. Based on threshold value, the dendrogram shows which queries will form a cluster. To get a more accurate representation, we can use a tighter (smaller) threshold. If we are more concerned of reducing the run time, we can choose a large threshold value. Several statistical software packages can generate the clustering diagrams. For example, we can use R [22], statistiXL [23], and SPSS [24].

D. Query Subset Selection

Once all the clusters among the queries are identified, one query is selected from each cluster to form the subset workload. A query can be select based on different criteria.

Execution time: The **first** method is to select a query based on the execution time. We can select the query with the lowest execution time. This helps to reduce the execution time of the subset workload. However, it can lead to a poor representation of the workload. Another approach is to select the query with the largest execution time among the queries of a cluster. This leads to a better representation of the workload, but increase the execution time of the subset workload. We can also select a query that has execution time close to the average execution time of the all queries of the cluster it belongs to. This approach has the advantage of representing all queries reasonably.

*Impact on workload ranking:*The **second** method is to select a query based on its *impact on the workload ranking*. We can select a query from a cluster which results in the smallest change in the overall workload ranking. This method can potentially results in preserving the bottlenecks within the workload. We can also select the query which has the smallest Euclidean distance to the other queries of that cluster. This method selects the query which represents the queries of the cluster in the best manner.

E. Selection of Workload and Parameters

The accuracy of the subset produced by SCRAP depends on workload and input parameters selected for identifying performance bottlenecks. In this section, we discuss how to select a workload and a set of input parameters.

Selection of a workload: Selection of a workload can be done by selecting a set of queries from a benchmark, or by collecting queries over a period of time using profile tools when the database system is running. A larger workload has a higher probability of capturing the important bottlenecks of the system, but at the cost of the extra time required for collecting data for the clustering. A database benchmark defines a workload in terms of query signatures, which are descriptions of the information which is needed from a database using specific structured query language (SQL) operators in an abstract form, allowing an application to specify the specific values. The specification for selecting input values of the queries is given in benchmark documentation. For example, *SELECT name FROM student_table WHERE gpa* $>$ *X*, is a query signature which gives the list of the students having GPA greater than

CPU	Two Intel XEON 2.0GHz w/ HT
Memory	4 X 512MB DDR DIMM
HDD	Hitachi Ultrastar, 73.5 G, 10,000 RPM

TABLE I
MACHINE DESCRIPTION

X, where X can be chosen within a range according to benchmark specification. How a query is executed inside a database depends on the values of the input parameters and availability of indexes on the input parameters. Although a query signature is the same, based on the selection of the input parameter values either sequential scan or index scan may be used by internal query executor. To select a workload based on a set of query signatures, multiple values of the input parameter should be used to accurately represent actual query workload. This issue is discussed further in detail in Section VI-B.

Selection of parameters: SCRAP produces the bottlenecks of the system with respect to the system parameters based on the impact of the parameters in system performance. All the parameters which are relevant for the database system under consideration must be included for finding the bottlenecks. Each query of workload is used to find the bottlenecks. Bottlenecks are used to cluster the queries. In order for the clustering to be accurate, the collection of the parameters needs to reflect all possible performance bottlenecks of the system. It is a good idea not to include the parameters which are not relevant. Irrelevant parameters increase the chance of finding inaccurate regression results due to aliasing [25], [26]. It is possible to use P&B design method to determine the relevant parameters, but we are not addressing this issue in this paper.

V. EXPERIMENTAL SETUP

For collecting experimental data, we use PostgreSQL 8.2 database system and the TPC-H benchmark. The description of the machine used for running experiments is given in Table I.

The TPC-H benchmark models a decision support system (DSS) [10]. For this experiment, the TPC-H database is populated by the data generation program *dbgen* using a scaling factor (SF) of one. A scaling factor of one corresponds to data size of 1GB. The TPC-H workload consists of 22 read only queries and two updates queries. For demonstration, we use only read-only queries. Queries are generated by using the query generation program *qgen*. We modify all 22 queries by adding EXPLAIN ANALYZE so that every query generates a query execution plan and total execution time, but does not generate any output results.

The PostgreSQL8.2 has approximately 100 configuration parameters [11]. For this demonstration, we only use those configuration parameters that are relevant to the read-only queries. Table II gives the high and low values for all 15 configuration parameter used in this demonstration. The values are selected according to recommendations made in the Postgres documentation [11]. The values of each parameter are chosen in a range such that it will act as monotonic parameter.

Parameter	**High Value**	**Low Value**
effective_cache_size (pages)	81920	8192
maintenance_work_mem (pages)	8192	1024
shared_buffers (pages)	16384	1024
temp_buffers (pages)	8192	1024
work_mem (KB)	8192	1024
random_page_cost (Relative to single sequential page cost)	2.0	5.0
cpu_tuple_cost (Relative to single sequential page cost)	0.01	0.03
cpu_index_tuple_cost (Relative to single sequential page cost)	0.001	0.003
cpu_operator_cost (Relative to single sequential page cost)	0.0025	0.0075
Checkpoint_timeout (seconds)	1800	60
deadlock_timeout (milliseconds)	60000	100
max_connections	5	100
fsync	true	false
geqo	true	false
stats_start_collector	false	true

TABLE II
THE POSTGRESQL8.2 CONFIGURATION PARAMETERS AND THEIR VALUES USED. EACH PAGE SIZE IS 8KB.

Although, `fsync` and `checkpoint_timeout` are not relevant for the read-only queries, but they are included to verify that our method is working correctly. The effect of these two parameters on the PostgreSQL performance will be very insignificant for the read-only queries. If their rankings/effects become low compared to other parameters, then it will give an indication that SCRAP identifies performance bottlenecks of the system in the correct order.

VI. RESULTS

The P&B effects of all configuration parameters are calculated using the *P&B design with foldover*. The resulting effects are shown in Table IV. In order to identify the insensitive parameters, a coefficient of variation(COV) of 0.05 is selected as the threshold. This value was selected based upon the query results. Through observation, queries with a COV less than 0.05 improperly identified bottlenecks, selecting parameters which have no effect upon performance, the variation of the parameter sensitives was often too low to accurately rank the parameters. The resulting cluster is shown in Table III.

Once the insensitive queries are removed from the workload the effects are normalized. Normalized to the sum of effects is shown in Table V, normalized to maximum effect is shown in Table VI, and the ranking is shown in Table VII.

To cluster the workload, a threshold for the Euclidean distance must be selected. In this analysis, the threshold is gradually incremented, and the subset workload ranking is compared of the workload ranking. The largest threshold which maintains a representation to the full workload ranking is selected. Once the queries are clustered, a query from that cluster is chosen to represent the cluster. For this experiment,

Query	**COV**	**max execution time**	**min execution time**
Q1	0.47	550132	189345
Q2	0.93	669063	45870
Q3	0.11	156749	101811
Q5	0.83	30.67	1.21
Q8	0.14	184459	108664
Q9	0.85	561527	71895
Q13	0.16	70561	44879
Q16	0.11	22017	15848
Q19	0.06	72690	63733
Q20	1.41	2140531	60710
Q21	0.22	1692037	831455
Q4	0.01	0.97	0.93
Q6	0.01	0.50	0.48
Q7	0.02	113730	105025
Q10	0.02	1.63	1.51
Q11	0.02	6372	5801
Q12	0.03	1.10	0.98
Q14	0.02	1.03	0.94
Q15	0.01	1.19	1.14
Q17	0.04	82384	74717
Q18	0.02	149872	136601
Q22	0.03	2629	2323

TABLE III
COEFFICIENT OF VARIATION (COV), MAXIMUM, MINIMUM OF EXECUTION TIMES FOR EACH QUERY. QUERIES WITH COV LESS THAN 0.05 ARE CONSIDERED AS INSENSITIVE TO PARAMETER VALUE CHANGES. THESE QUERIES ARE GROUPED TOGETHER IN THE BOTTOM HALF OF THE TABLE.

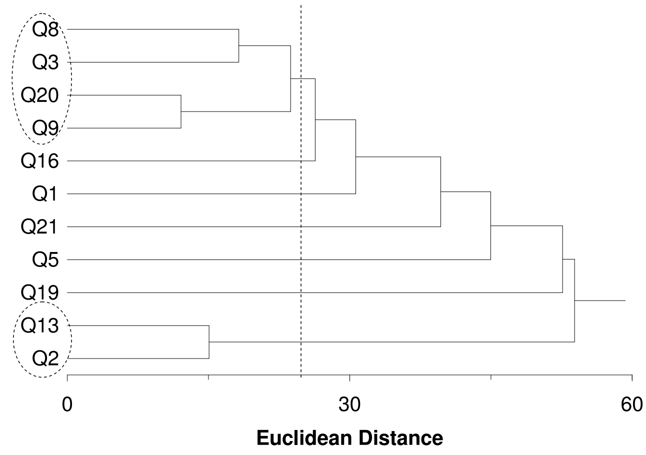

Fig. 1. DENDROGRAM OF THE QUERIES BASED ON SIMILARITIES AMONG EUCLIDEAN DISTANCE OF VECTORS FORMED BY NORMALIZED EFFECTS WITH RESPECT TO THE SUM OF EFFECTS. CIRCLE DENOTES CLUSTER AND VERTICAL LINE DENOTES THRESHOLD.

the query which results in the smallest impact upon the workload ranking is selected, under the assumption that the workload ranking would be best represented if it was as close as possible to the full workload. This method is performed by incrementally clustering a query, testing the clustering options, and selecting the query which has the lowest impact upon overall ranking.

In the rest of this section, we will compare the three methods previously discussed in Section IV-C. First, the bottlenecks produced by the subset are evaluated by different clustering techniques against the original workload bottlenecks. Second, we use the subset to predict the execution time of the full workload. Finally, this section ends with a comparison between two workloads based upon the same query signature.

A. Clustering Validation

In this section, we estimate the quality of the representational workload produced by SCRAP. We compare the clustering due to the Euclidean distance of vectors produced by three different methods: normalization with respect to the sum of effects, normalization with respect to max effects, and ranking. In addition to these three methods, we form clustering by randomly selecting queries to verify the clustering has a dependence upon the clusters chosen. To validate the representational workload, two additional analysis are performed. The P&B design will be applied to a workload and the representational workload. The full workload ranking will then be compared to the representational workload. Next, using the representational workload, the overall execution time of the full workload is estimated.

Figures 1, 2, and 3 show the dendrograms of queries based on the Euclidean distance between the vectors formed by normalizing with respect to the sum of effects, normalized with respect to maximum effects, and ranking, respectively. Initially a threshold is selected in the dendrogram and gradually incremented until the estimated workload ranking deviates from the complete workload ranking. In the dendrograms, y-axis represents the queries and x-axis represents the similarity among queries based on Euclidean distance. For sensitive queries in Figure 1, there are two clusters {Q3, Q8, Q9, Q20} and {Q2, Q13} based on threshold Euclidean distance of 15. In Figure 2, there are three clusters {Q1, Q8, Q16}, {Q19, Q21}, and {Q2, Q13} based on a threshold Euclidean distance of 1.2. In Figure 3, there are three clusters {Q2, Q13, Q19} and {Q3, Q8} based a threshold Euclidean distance of 17.

Normalizing with respect to both the sum and maximum clustered four queries, while the ranking method was only able to cluster three queries before the ranking was adversely effected.

To validate the workload bottlenecks, P&B was run upon the entire workload, a subset of these queries was used to estimate the complete workload's ranking. If the representational workload displays the same ranking as the unclustered workload, the bottleneck of the system is represented properly; an indicator that the ranking method was performing correctly. A random clustering was used to test the value of clustering by bottlenecks.

To rank the workload, the parameter sensitivities are summed for each query. The summed parameter are then ranked. When a subset of queries is used, weighting is needed. If a query represents M number of queries, it is given a weighting of M, to ensure that the cluster is still given equal representation for the full workload. When queries are clustered, the query that is selected to remain is assigned a

Parameter	Q1	Q2	Q3	Q5	Q8	Q9	Q13	Q16	Q19	Q20	Q21
Checkpoint_timeout	63139	28711	11325	57.130	8910	602068	4130	1590	20594	2388937	207883
deadlock_timeout	11923	5582	16801	1.004	64500	54715	5214	1546	21605	2356317	100185
fsync	59809	13556	20707	0.705	4534	274194	3424	1329	22334	2399193	31385
max_connections	97626	68939	34326	56.675	77876	628112	3678	1792	16788	2364189	117188
shared_buffers	162386	2459527	25049	1.696	34787	770430	132957	5997	139032	2450275	2632182
stats_start_collector	71420	28909	22561	1.142	25181	328104	4278	2122	17883	2376664	179365
cpu_index_tuple_cost	10651	103431	11380	0.808	42106	149179	2460	1697	16344	2381224	657371
cpu_operator_cost	81071	130508	57606	56.299	46111	349389	480	1714	41086	3664134	15003
cpu_tuple_cost	58019	48172	31252	56.356	74791	48262	5808	1836	50802	2370930	398057
effective_cache_size	75846	6874535	43373	56.156	11954	304263	238544	979	27854	2407511	627016
geqo	3858	30488	56499	56.704	82480	529011	853	1679	18327	2377343	141108
maintenance_work_mem	75774	74429	26012	0.747	127031	780976	1351	1048	10422	2397854	413680
random_page_cost	8693	46948	17519	397.908	43699	1694532	3447	2145	6969	1127905	490531
temp_buffers	101473	42060	45193	0.861	31217	571133	2713	1238	11522	2391281	240543
work_mem	5523703	13295	183039	56.548	359544	142035	24839	63760	3027	2370169	310306

TABLE IV
THE P&B EFFECTS OF ALL CONFIGURATION PARAMETERS FOR TPC-H QUERIES WHICH ARE SENSITIVE TO PARAMETER VALUE CHANGES.

Parameter	Q1	Q2	Q3	Q5	Q8	Q9	Q13	Q16	Q19	Q20	Q21
Checkpoint_timeout	0.99	0.29	1.88	7.13	0.86	8.33	0.95	1.76	4.85	6.67	3.17
deadlock_timeout	0.19	0.06	2.79	0.13	6.23	0.76	1.20	1.71	5.09	6.58	1.53
fsync	0.93	0.14	3.44	0.09	0.44	3.79	0.79	1.47	5.26	6.70	0.48
max_connections	1.52	0.69	5.70	7.08	7.53	8.69	0.85	1.98	3.95	6.60	1.79
shared_buffers	2.54	24.67	4.16	0.21	3.36	10.66	30.62	6.63	32.75	6.84	40.11
stats_start_collector	1.11	0.29	3.74	0.14	2.43	4.54	0.99	2.35	4.21	6.63	2.73
cpu_index_tuple_cost	0.17	1.04	1.89	0.10	4.07	2.06	0.57	1.88	3.85	6.65	10.02
cpu_operator_cost	1.27	1.31	9.56	7.03	4.46	4.83	0.11	1.89	9.68	10.23	0.23
cpu_tuple_cost	0.91	0.48	5.19	7.04	7.23	0.67	1.34	2.03	11.96	6.62	6.07
effective_cache_size	1.18	68.96	7.20	7.01	1.16	4.21	54.94	1.08	6.56	6.72	9.56
geqo	0.06	0.31	9.38	7.08	7.97	7.32	0.20	1.86	4.32	6.64	2.15
maintenance_work_mem	1.18	0.75	4.32	0.09	12.28	10.81	0.31	1.16	2.45	6.69	6.30
random_page_cost	0.14	0.47	2.91	49.69	4.22	23.45	0.79	2.37	1.64	3.15	7.48
temp_buffers	1.58	0.42	7.50	0.11	3.02	7.90	0.62	1.37	2.71	6.68	3.67
work_mem	86.24	0.13	30.37	7.06	34.75	1.97	5.72	70.47	0.71	6.62	4.73

TABLE V
THE P&B EFFECTS OF ALL CONFIGURATION PARAMETERS NORMALIZED TO THE SUM OF EFFECTS.

weighting. Calculating the estimated ranking is accomplished by multiplying the query's sensitivity by the weighting, and summing the weighted effects.

The representational workload ranking is compared to the workload in Table VIII. In all three methods of forming clusters, the primary bottlenecks remain the same. Both of the normalizing techniques had a consistent ranking, the main eight bottlenecks remain unchanged with four queries clustered. The ranking method has a fairly consistent workload ranking, it maintained the top six parameters when the exception of swapping the fourth/fifth parameter with only three queries are clustered.

The randomly clustered queries have a poor representation of the full workload according to the result in Table IX. The top five parameters maintained their ranking, but the order of the ranking has shifted when only two queries are clustered. It is apparent that SCRAP's clustering is able to cluster four queries without significantly changing the query ranking.

The execution time of the entire workload is estimated using the subsets produced by three different clustering methods. In order to validate that the queries are representative of the entire workload, the subset of the workload is used to predict the workload under the default case, as well as a tuned case using a new set of values for three parameters. The values are given in Table X.

Rank	Before clustering	After clustering
1	work_mem	work_mem
2	effective_cache_size	shared_buffers
3	shared_buffers	effective_cache_size
4	cpu_operator_cost	random_page_cost
5	random_page_cost	cpu_operator_cost
6	temp_buffers	geqo
7	maintenance_work_mem	temp_buffers
8	deadlock_timeout	deadlock_timeout
9	geqo	maintenance_work_mem
10	cpu_tuple_cost	cpu_tuple_cost
11	checkpoint_timeout	checkpoint_timeout
12	fsync	max_connections
13	cpu_index_tuple_cost	cpu_index_tuple_cost
14	max_connections	fsync
15	stats_start_collector	stats_start_collector

TABLE IX
WORKLOAD RANKING FOR RANDOMLY SELECTED CLUSTERS.

Parameter	Q1	Q2	Q3	Q5	Q8	Q9	Q13	Q16	Q19	Q20	Q21
Checkpoint_timeout	0.008	0.001	0.011	0.143	0.010	0.799	0.000	0.047	0.009	0.595	0.046
deadlock_timeout	0.004	0.004	0.197	0.003	0.160	0.770	0.015	0.034	0.006	0.594	0.127
fsync	0.004	0.003	0.047	0.004	0.018	0.884	0.006	0.032	0.009	0.605	0.047
max_connections	0.007	0.017	0.244	0.141	0.173	0.301	0.007	0.042	0.002	0.603	0.069
shared_buffers	0.010	0.345	0.018	0.006	0.072	0.267	0.573	0.114	1.000	0.636	1.000
stats_start_collector	0.004	0.013	0.093	0.005	0.015	0.272	0.017	0.052	0.003	0.599	0.082
cpu_index_tuple_cost	0.008	0.016	0.175	0.004	0.095	0.466	0.011	0.037	0.002	0.598	0.213
cpu_operator_cost	0.004	0.032	0.383	0.141	0.073	1.000	0.021	0.035	0.009	1.000	0.018
cpu_tuple_cost	0.005	0.012	0.141	0.140	0.136	0.501	0.010	0.038	0.008	0.599	0.128
effective_cache_size	0.003	1.000	0.282	0.139	0.033	0.980	1.000	0.022	0.001	0.604	0.142
geqo	0.003	0.014	0.411	0.142	0.221	0.378	0.005	0.041	0.006	0.599	0.073
maintenance_work_mem	0.009	0.017	0.326	0.004	0.288	0.387	0.002	0.033	0.012	0.615	0.232
random_page_cost	0.011	0.001	0.173	1.000	0.143	0.809	0.010	0.057	0.003	0.206	0.046
temp_buffers	0.015	0.004	0.280	0.004	0.056	0.896	0.007	0.030	0.002	0.603	0.059
work_mem	1.000	0.011	1.000	0.141	1.000	0.465	0.097	1.000	0.001	0.605	0.022

TABLE VI
THE P&B EFFECTS OF ALL CONFIGURATION PARAMETERS NORMALIZED TO THE MAXIMUM EFFECT.

Parameter	Q1	Q2	Q3	Q5	Q8	Q9	Q13	Q16	Q19	Q20	Q21
Checkpoint_timeout	15	2	14	15	11	3	2	2	1	4	1
deadlock_timeout	1	15	1	8	1	14	12	1	15	14	15
fsync	15	1	7	8	15	11	12	15	11	4	5
max_connections	15	15	7	15	2	3	15	15	13	4	5
shared_buffers	15	15	14	8	6	7	1	15	11	14	15
stats_start_collector	15	15	7	8	11	11	15	15	3	1	15
cpu_index_tuple_cost	15	15	7	8	6	15	15	15	3	14	8
cpu_operator_cost	15	15	2	8	6	7	12	15	11	14	15
cpu_tuple_cost	15	15	14	15	11	14	12	15	11	14	5
effective_cache_size	15	15	14	1	15	1	12	15	14	15	8
geqo	15	15	14	15	6	14	12	15	11	14	15
maintenance_work_mem	15	15	14	15	11	11	3	15	11	14	8
random_page_cost	15	15	15	8	15	4	12	15	11	14	5
temp_buffers	15	15	7	15	11	7	12	15	12	14	15
work_mem	15	15	14	15	15	11	12	15	11	14	15

TABLE VII
RANKING OF THE ALL CONFIGURATION PARAMETERS USING ROUNDING NORMALIZED EFFECT WITH RESPECT TO MAXIMUM EFFECT UP TO THE FIRST DECIMAL POINT METHOD.

Parameter	New value
effective_cache_size	12288
shared_buffers	12288
work_mem	4096

TABLE X
PARAMETER VALUE USED FOR TUNING THE SYSTEM.

	Default value	New value	%Δ
full workload	3219	2973	
subset (Normalized to Max)	3207	2978	0.18%
subset (Normalized to Sum)	2646	2563	18.81%
subset (Rank)	3465	3182	7.33%
subset (Random Cluster)	4514	4331	42.94%

TABLE XI
ESTIMATION OF EXECUTION TIME IN MILLISECONDS USING THE SUBSET QUERIES.

To estimate the execution time of the entire workload, the workload was weighted. The weighting is calculated by taking the average execution time (obtained in the P&B experiment) of all queries that are represented by that query divided by that queries' execution time.

For both the default case and the tuned case, the execution times were collected five times and averaged. The default case uses the default parameter values given by PostgreSQL. The resulting data is given in Table XI. The estimation for the normalization to the maximum effect estimated the total execution time within 0.18%. The normalization to the maximum effect clearly gives the most effective representational workload, giving the most consistent workload ranking and estimating the execution time the most accurately.

B. Query Workload Comparison

In a database system, how a query will be executed internally depends on the input parameters used in the corresponding query signature. A constant query signature does not mean the database will always process the query in the same manner. Query signature contains a placeholder for the input parameter. The input parameter can be set to random values according to the workload documentation or the application of interest. The random values can have drastic effects on a query execution plan. For example, if a query requests the list of the names of university students over 15 years old, the database query

	Normalized to Sum		**Normalized to Max**		**Ranking by Rounding**	
Rank	Before clustering	After clustering	Before clustering	After clustering	Before	After
1	work_mem	work_mem	work_mem	work_mem	work_mem	work_mem
2	shared_buffers	shared_buffers	effective_cache_size	effective_cache_size	shared_buffers	shared_buffers
3	effective_cache_size	effective_cache_size	shared_buffers	shared_buffers	effective_cache_size	effective_cache_size
4	random_page_cost	random_page_cost	cpu_operator_cost	cpu_operator_cost	cpu_operator_cost	cpu_operator_cost
5	cpu_operator_cost	maintenance_work_mem	random_page_cost	random_page_cost	random_page_cost	maintenance_work_mem
6	maintenance_work_mem	geqo	temp_buffers	temp_buffers	maintenance_work_mem	random_page_cost
7	geqo	max_connections	maintenance_work_mem	maintenance_work_mem	geqo	geqo
8	cpu_tuple_cost	cpu_operator_cost	deadlock_timeout	deadlock_timeout	temp_buffers	deadlock_timeout
9	max_connections	cpu_tuple_cost	geqo	cpu_index_tuple_cost	deadlock_timeout	max_connections
10	deadlock_timeout	deadlock_timeout	cpu_tuple_cost	geqo	max_connections	cpu_index_tuple_cost
11	cpu_index_tuple_cost	cpu_index_tuple_cost	checkpoint_timeout	cpu_tuple_cost	cpu_index_tuple_cost	cpu_tuple_cost
12	temp_buffers	checkpoint_timeout	fsync	checkpoint_timeout	cpu_tuple_cost	temp_buffers
13	checkpoint_timeout	temp_buffers	cpu_index_tuple_cost	fsync	stats_start_collector	stats_start_collector
14	fsync	stats_start_collector	max_connections	max_connections	checkpoint_timeout	checkpoint_timeout
15	stats_start_collector	fsync	stats_start_collector	stats_start_collector	fsync	fsync

TABLE VIII
WORKLOAD RANKING USING THREE DIFFERENT VECTORS FORMED BY EFFECT NORMALIZED TO SUM, EFFECT NORMALIZED TO MAX, AND RANKING BY ROUNDING METHODS.

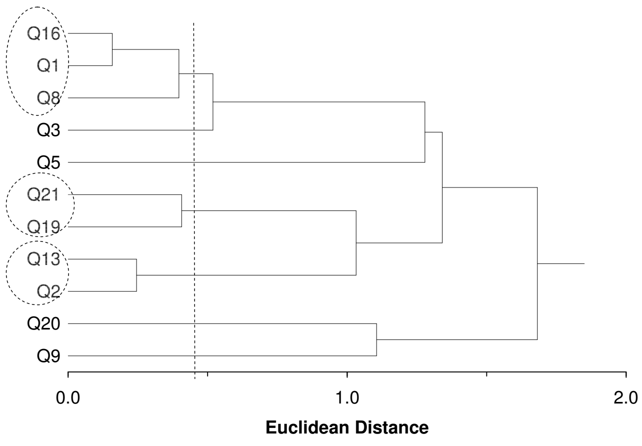

Fig. 2. DENDROGRAM OF THE QUERIES BASED ON SIMILARITIES AMONG EUCLIDEAN DISTANCE OF VECTORS FORMED BY THE NORMALIZED EFFECTS WITH RESPECT TO THE MAXIMUM EFFECT. CIRCLE DENDOTES CLUSTER AND VERTICAL LINE DENOTES THRESHOLD.

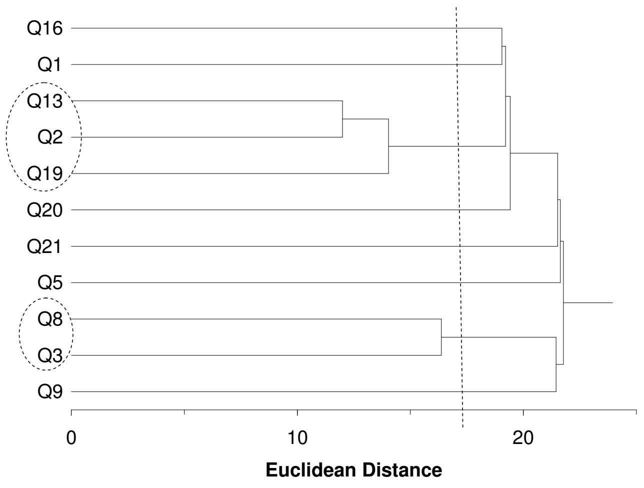

Fig. 3. DENDROGRAM OF THE QUERIES BASED ON SIMILARITIES AMONG EUCLIDEAN DISTANCE OF VECTORS FORMED BY RANK OF THE PARAMETERS. CIRCLE DENOTES CLUSTER AND VERTICAL LINE DENOTES THRESHOLD.

executor will choose to use a sequential scan irrespective of whether an index of ages exists or not, as most of the student's ages qualify this condition. However, if a query requests a list of the name students whose ages are over 60, the database will use an index on age. To find how random values in same query signature will affect clustering, two sets using the same query signatures are compared.

Table XII gives the normalized ranking of two sets of the same queries of data. For illustration, only queries Q1 and Q3 are shown. This does not mean that all of the queries showed different ranking; Q5 and Q13 had very similar ranking as shown in Table XIII. This implies that a set of query signatures can not be completely characterized. The more values tested for the query signatures, the more accurately the query signature will be represented in the workload. If it is decided to cluster a workload with 22 query signatures, and to test five values for each query signature, SCRAP would be executed on all 110 queries.

Parameter	**S1Q3**	**S2Q3**	Δ	**S1Q9**	**S2Q9**	Δ
checkpoint_timeout	0.01	0.02	0.01	0.80	0.17	0.63
deadlock_timeout	0.20	0.00	0.19	0.77	0.19	0.58
fsync	0.05	0.10	0.05	0.88	0.01	0.88
max_connections	0.24	0.10	0.14	0.30	0.21	0.09
shared_buffers	0.02	0.30	0.28	0.27	0.45	0.18
stats_start_collector	0.09	0.18	0.09	0.27	0.13	0.14
cpu_index_tuple_cost	0.17	0.10	0.07	0.47	0.02	0.45
cpu_operator_cost	0.38	0.02	0.37	1.00	0.05	0.95
cpu_tuple_cost	0.14	0.24	0.10	0.50	0.09	0.41
effective_cache_size	0.28	0.01	0.27	0.98	0.01	0.97
geqo	0.41	0.02	0.39	0.38	0.23	0.15
maintenance_work_mem	0.33	0.09	0.24	0.39	0.46	0.08
random_page_cost	0.17	0.02	0.15	0.81	1.00	0.19
temp_buffers	0.28	0.10	0.18	0.90	0.07	0.83
work_mem	1.00	1.00	0.00	0.46	0.19	0.28

TABLE XII
COMPARISON OF QUERIES WITH THE SAME QUERY SIGNATURE AND DIFFERENT SENSITIVITIES.

VII. CONCLUSION

The current rapid growth of data forces businesses to build larger databases. Larger the databases are the more difficult to tune due to the large number of queries, the

Parameter	S1Q5	S2Q5	Δ	S1Q13	S2Q13	Δ
checkpoint_timeout	0.14	0.14	0.00	0.00	0.02	0.02
deadlock_timeout	0.00	0.00	0.00	0.02	0.02	0.01
fsync	0.00	0.00	0.00	0.01	0.02	0.01
max_connections	0.14	0.15	0.00	0.01	0.01	0.01
shared_buffers	0.01	0.00	0.01	0.57	0.56	0.01
stats_start_collector	0.01	0.00	0.00	0.02	0.02	0.00
cpu_index_tuple_cost	0.00	0.00	0.00	0.01	0.01	0.00
cpu_operator_cost	0.14	0.14	0.00	0.02	0.00	0.02
cpu_tuple_cost	0.14	0.14	0.00	0.01	0.03	0.02
effective_cache_size	0.14	0.14	0.00	1.00	1.00	0.00
geqo	0.14	0.14	0.00	0.01	0.00	0.00
maintenance_work_mem	0.00	0.00	0.00	0.00	0.01	0.01
random_page_cost	1.00	1.00	0.00	0.01	0.01	0.00
temp_buffers	0.00	0.00	0.00	0.01	0.02	0.01
work_mem	0.14	0.14	0.00	0.10	0.11	0.01

TABLE XIII
COMPARISON OF QUERIES WITH THE SAME QUERY SIGNATURE AND SIMILAR SENSITIVITIES.

large execution times of workloads, and the difficulty of creating a test workload that represents the real workload of the system. SCRAP has potential as a method to reduce the high cost of database tuning while improving the confidence that the primary database bottlenecks are tested. SCRAP is presented as a method for creating a compact representational workload. SCRAP clusters queries based on the similarities among their performance bottleneck parameters. Our results show SCRAP reduces the 22 read-only TPC-H queries down to eight. The subset of queries has been shown to estimate the performance bottlenecks with accuracy, and estimate the overall execution time with only a 0.18% error. We find that the same query signature produces different peformance bottlenecks depending on the input parameter values. This suggests that to correctly characterize a workload consisting of query signatures, multiple values for each parameter should be used in a manner such that these values estimate the entire search space for that query signature.

VIII. ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation grants CCF-0621462 and CCF-0541162, the University of Minnesota Digital Technology Center Intelligent Storage Consortium, and the Minnesota Supercomputing Institute.

REFERENCES

[1] B. Dageville and K. Dias, “Oracle’s Self-Tuning Architecture and Solutions,” in *IEEE Data Engineering Bulletin*, vol. 29, no. 3, 2006, pp. 24–31.

[2] S. Agrawal, N. Bruno, S. Chaudhuri, and V. Narasayya, “AutoAdmin: Self-Tuning Database SystemsTechnology,” in *IEEE Data Engineering Bulletin*, vol. 29, no. 3, 2006, pp. 7–15.

[3] S. Lightstone, G. Lohman, P. Haas, V. Markl, J. Rao, A. Storm, M. Surendra, and D. Zilio, “Making DB2 Products Self-Managing: Strategies and Experiences,” in *IEEE Data Engineering Bulletin*, vol. 29, no. 3, 2006, pp. 16–23.

[4] K. Keeton and D. Patterson, “Towards a Simplified Database Workload for Computer Architecture Evaluations,” in *Workload Characterization for Computer System Design, edited by L. K. John and A. A. Maynard, Kluwer Academic Publishers,*, 2000.

[5] S. Chaudhuri and G. Weikum, “Rethinking database architecture: Towards s self-tuning risc-style database system,” in *VLDB*, 2000, pp. 1–10.

[6] K. Dias, M. Ramacher, U. Shaft, V. Venkataramamani, and G. Wood, “Automatic Performance Diagnosis and Tuning in Oracle,” in *CIDR*, 2005, pp. 1110–1121.

[7] G. Group, “The total cost of ownership: The impact of system management tools,” 1996.

[8] H. Group, “Achieving faster time-to-benefit and reduced tco with oracle certified configurations,” March, 2002.

[9] R. Plackett and J. Burman, “The Design of Optimum Multifactorial Experiments,” in *Biometrika Vol. 33 No. 4*, 1946, pp. 305–325.

[10] Transaction Processing Council. [Online]. Available: http://www.tpc.org/

[11] “PostgreSQL DBMS Documentation.” [Online]. Available: http://www.postgresql.org/

[12] A. Ailamaki, D. DeWitt, M. Hill, and D. Wood, “DBMSs on a Modern Processor: Where Does Time Go?” in *The VLDB Journal*, 1999, pp. 266–277.

[13] M. Shao, A. Ailamaki, and B. Falsafi, “DBmbench: Fast and Accurate Database Workload Representation on Modern Microarchitecture,” in *CASCON*, 2005, pp. 254–267.

[14] L. Barroso, K. Gharachorloo, and E. Bugnion, “Memory System Characterization of Commercial Workloads,” in *ISCA*, 1998, pp. 3–14.

[15] P. Ranganathan, K. Gharachorloo, S. V. Adve, and L. A. Barroso, “Performance of Database Workloads on Shared-Memory Systems with Out-of-Order Processors,” in *ASPLOS*, 1998, pp. 307–318.

[16] P. Trancoso, J.-L. Larriba-Pey, Z. Zhang, and J. Torrellas, “The memory performance of dss commercial workloads in shared-memory multiprocessors,” in *HPCA*, 1997.

[17] P. Trancoso, C. Adamou1, and H. Vandierendonck, “Reducing TPC-H Benchmarking Time,” in *PCI*, 2005.

[18] H. Vandierendonck and P. Trancoso, “Building and Validating a Reduced TPC-H Benchmark,” in *MASCOTS*, 9 2006, pp. 383–392.

[19] J. Yi, D. Lilja, and D. Hawkins, “A statistically rigorous approach for improving simulation techniques,” in *HPCA*, 2003.

[20] D. Lilja, *Measuring Computer Performance: A Practitioner‘s Guide*. Cambridge University Press, 2000.

[21] D. Montgomery, *Design and Analysis of Experiments*. Wiley, 2001.

[22] “The R Project for Statistical Computing.” [Online]. Available: http://www.r-project.org/

[23] “statistiXL.” [Online]. Available: http://www.statistixl.com/

[24] “SPSS.” [Online]. Available: http://www.spss.com/

[25] M. Hamada and C. Wu, “Analysis of Censored Data from Highly Fractionated Experiments,” in *Technometrics*, vol. 33, no. 1, 1991, pp. 25–38.

[26] O. Samset, “Identifying Active Factors from Non-Geometric Plackett-Burman Designs and Their Half-Fractions,” in *Technical Report, Department of Mathematical Sciences. The Norwegian University of Science and Technology*, 1999.