

# SARD: A Statistical Approach for Ranking Database Tuning Parameters

Biplob K. Debnath, James Skarie, David J. Lilja, and Mohamed F. Mokbel  
University of Minnesota, Twin-Cities, USA.  
E-mail: {debna004, skar0059, lilja}@ece.umn.edu , mokbel@cs.umn.edu

## Abstract

Traditionally, DBMSs are shipped with hundreds of configuration parameters. To address a broad class of applications, such configuration parameters are set to default values. Since the database performance highly depends on the appropriate settings of the configuration parameters, DBAs spend a lot of their time and effort to find the best parameters values for tuning the performance of the application of interest. In many cases, they rely on their experiences and some rules of thumbs. However, as heuristics are used, time may be wasted by tuning those parameters which may have no or marginal effects. Moreover, tuning effects also vary depending on the expertise of the DBAs, but skilled DBAs are increasingly becoming rare and expensive to employ. To address these problems, we present a *Statistical Approach for Ranking Database parameters (SARD)*, which is based on the Plackett & Burman statistical design methodology. SARD takes the query workload and the number of configuration parameters as inputs, and using only a linear number of experiments, generates a ranking of database parameters based on their relative impacts on the DBMS performance. Experimental results using TPC-H and PostgreSQL show that SARD generated ranking can correctly identify critical configuration parameters.

## 1 Introduction

Businesses are increasingly building larger databases to cope with the rapid current growth of data. Consistent performance of the underlying database system is a key to success of a business. Typical database management system (DBMS) has hundreds of configuration parameters and the appropriate setting of these parameters play a critical role in performance. DBMSs are shipped with default values for all configuration parameters targeting a wide range of applications. Database administrators (DBAs) are expected to tune the performance of these generic DBMSs to the application of their interest. The success of tuning depends on many factors including the query workload, relational schemas, as well as the expertise of the DBAs [29]. However, skilled DBAs are becoming increasingly rare and expensive [22]. A recent study on information technology ver-

sus DBA costs showed that personnel cost is the largest category of the total cost, estimated at 47% of the total cost of ownership [15]. Many DBAs spend nearly a quarter of their time on tuning activities [29]. To reduce the total cost of ownership, DBAs can spend their time and effort in tuning only those configuration parameters which have the most impact on system performance.

A sound statistical methodology for finding the appropriate values of configuration parameters is to perform a *sensitivity analysis*. The major problem of performing a sensitivity analysis in a DBMS is the large number of configuration parameters which can assume multiple values. For example, PostgreSQL has approximately 100 configuration parameters and all parameters can assume at least two possible values [2]. Even if each configuration parameter assumes only two values, we have to perform  $2^{100}$  experiments to do a complete sensitivity analysis, which is not feasible in terms of time and effort. To overcome this problem, in many cases DBAs rely on their experiences and rules of thumb to select the appropriate configuration parameters for tuning. Nonetheless, as heuristics are often used, time and effort may be wasted to enhance the performance by tuning those parameters which may have no or marginal effects on overall performance. Misdirected tuning efforts increase the total cost of ownership [11, 14, 16, 17]. A ranking of the parameters based on their impact on system performance will greatly help DBAs to prioritize their tuning tasks. To the best of our knowledge, there is no study which statistically provides a ranking of the configuration parameters based on their impact on DBMS performance.

In this work, using a *design of experiments* based PLACKETT & BURMAN (P&B) methodology [28], we present a Statistical Approach for Ranking Database configuration parameters (SARD). SARD can be used to discard unimportant tuning parameters which have marginal or no impact on DBMS performance. In particular, SARD 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 relative ranking of the parameters based on their impact on performance.* 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 us-

ing profiling tools. Our objective is to find a solution which is feasible in terms of time and effort.

The main idea of SARD is to conduct a set of experiments such that these experiments will provide an approximate sampling of the entire search space. In each experiment, parameter values are varied systematically over a specified range of acceptable values. Subsequent analyses of the collected experimental data are used to estimate the effects of configuration parameters on system performance. To estimate the effects of all input parameters and all of their interactions, we can use a *full factorial design* such as ANOVA, in which the system response is measured for all possible input combinations [23]. However, a *full factorial design* requires an exponential number of experiments. To solve this problem, SARD uses the P&B *two-level factorial design* methodology based on the following assumptions: 1) stimulating the system with monotonically non-decreasing parameters at their extreme values will provoke the greatest response for each parameter; and 2) only single and two-factor parameters interactions are need to be considered. Adopting P&B design method helps us to reduce the required number of experiments from exponential to linear.

SARD is a generic methodology which can also be applied to the non-database systems. Here, we demonstrate that SARD is able to find performance bottleneck parameters using the PostgreSQL [2] DBMS and the TPC-H benchmark [3]. Our contributions in this paper are the following:

- *A methodology for ranking configuration parameters.* All configuration parameters are ranked based on their relative impacts on DBMS performance for each query of the workload. Individual ranking of the parameters for all queries will be combined to estimate the final ranking of all configuration parameters for the entire workload.
- *A methodology for classifying queries.* All queries of a workload are divided into two groups based on their performance sensitivity to the configuration parameters.
- *Providing experimental evidence.* Using TPC-H read only queries and PostgreSQL DBMS, we experimentally demonstrate that our methodology can effectively rank performance bottleneck parameters.

The remainder of the paper is organized as follows: Section 2 describes the related work on DBMS performance tuning; Section 3 describes our experimental methodology and a brief overview of the statistical P&B design; Section 4 describes some guidelines for using P&B design for database systems and how to apply it in a DBMS for ranking the configuration parameters based on their impact on the system; Section 5 describes the experimental setup; Section 6 explains our results; finally Section 7 concludes the discussion.

## 2 Related Work

Recent research on database tuning can be classified into three broad categories: tuning the *physical design*, *identifying performance bottlenecks*, and *tuning the configuration parameters*. SARD falls into both the second and third categories.

**Tuning the Physical Design:** Major database vendors offer tools for automating database physical design. For example, Oracle 10g provides tools for selection of indexes, materialized views, for identifying the root causes of performance bottlenecks, and for estimating the benefit of eliminating a performance bottleneck [1, 12–14]. Microsoft SQL Server provides tools that allow for integrated selection of indexes, indexed views, and horizontal partitions [4–10]. IBM’s DB2 recommends indexes, materialized views, shared nothing partitions, and multidimensional clustering of tables [18, 21, 22, 24, 32, 33, 36].

**Identifying Performance Bottlenecks:** Rule-based decision trees are used to identify the potential sources of performance bottleneck [25]. A decision tree is formed based on a set of rules for finding the bottlenecks. ADDM uses a common performance metric ‘database time’ [14]. ADDM posses a holistic view of the database, identifies root causes of the performance bottlenecks, and estimates the benefits of eliminating performance bottlenecks. However, ADDM ignores system configuration parameters settings. A *design of experiments* based approach is used to evaluate the statistical significance of configurable parameters, the interaction effects between each parameter, web function types, and to rank key configurable system parameters that significantly impact overall system performance for E-Commerce systems [30, 31]. The drawback is that this an ad-hoc approach, lacking sound statistical methodology. Also, there is no upper bound on numbers of experiments needed to collect necessary data for determining rank.

**Selecting Configuration Parameters:** IBM DB2 UDB provides a wizard for automatically selecting the initial values for the configuration parameters [20]. Configuration choices are made by modeling each database configuration setting as a mathematical expression consisting of user specification of the database environments, automatically sensed system characteristics, and expert heuristics. Nonetheless, this feature suffers from the following problems: as the user specified inputs can have very different characteristics than the built-in workload models, the recommended values can be inaccurate; due to the use of heuristics, there is no statistical evidence that these values are correct; and no ranking is provided. In this work, we will address all these problems. As our conclusions are based on the real workloads, the first problem will not occur at all. The third problem is solved using a sound statistical methodology, which will indirectly take care of the second problem.

SARD uses P&B design for estimating the impact of a configuration parameter on DBMS performance for a par-

ticular workload. The P&B design has been used in various applications, such as improving the simulation methodology of micro architecture research [35]; identifying the most significant processor parameters, selecting a subset of benchmarks, analyzing the effect of an enhancement on processor performance [35]; characterizing and comparing existing computer architecture simulation techniques [34]; to systematically stressing statistical simulation by creating different performance bottlenecks [19]; and identifying the performance critical buses of a micro architecture [27]. To the best of our knowledge, SARD is the first to apply the statistical design of experiments based P&B design approach to study a DBMS.

### 3 Design of Experiments Based Methodology

SARD is a *design of experiments* based approach. The major goal of the *design of experiments* is to gather the maximum information about a system with minimum effort [23]. Experiments are conducted based on a given specification to collect information about system performance. The subsequent analysis of resulting experimental data is used to identify the important factors (parameters), and the presence of interactions between the factors. Given  $N$  parameters, in total there are  $(2^N - 1)$  effects to be estimated. The simplest design strategy is to use *full factorial design* for example ANOVA, in which system response is measured for all possible input combinations [23]. However, it requires exponential number of experiments on number of parameters. Clearly, for DBMS which typically has many parameters, this is not a feasible approach at all. For example, PostgreSQL has approximately 100 configuration parameters. If each parameter takes only two values, a *full factorial design* will require to conduct  $2^{100}$  experiments.

To reduce the number of experiments, SARD makes a few assumptions. First assumption, for each parameter SARD considers only 2 values: minimum and maximum. The intuition behind this is that stimulating the system with inputs at their extreme values will provoke the maximum range of output responses for each input. A second related assumption is that the provoked response, such as the total execution time, is a monotonic function of the input parameter values. We provide some guidelines in Section 4.4 of dealing with the case where this assumption is not valid. Third assumption is based on *sparsity of effects principle*: system response is largely dominated by a few main factors and low-order interactions; the effect of higher order interactions on response is not statistically significant. As a consequence, we can safely ignore the effects of higher order interactions. Based on these assumptions, SARD uses a *two-level factorial* design named Plackett & Burman (P&B) design [28], which requires only linear number of experiments compared to the exponential number of experiments required by *full-factorial design*.

For each experiment of the P&B design, the value of each parameter is given by prescribed *design matrix*. An example of a design matrix is given by first seven columns

A	B	C	D	E	F	G	Execution Time		
							Q1	Q2	Q3
+1	+1	+1	-1	+1	-1	-1	34	110	10.2
-1	+1	+1	+1	-1	+1	-1	19	72	10.1
-1	-1	+1	+1	+1	-1	+1	111	89	10.3
+1	-1	-1	+1	+1	+1	-1	37	41	10.3
-1	+1	-1	-1	+1	+1	+1	61	96	10.2
+1	-1	+1	-1	-1	+1	+1	29	57	10.2
+1	+1	-1	+1	-1	-1	+1	79	131	10.3
-1	-1	-1	-1	-1	-1	-1	19	47	10.1
-1	-1	-1	+1	-1	+1	+1	135	107	10.3
+1	-1	-1	-1	+1	-1	+1	56	74	10.3
+1	+1	-1	-1	-1	+1	-1	112	48	10.1
-1	+1	+1	-1	-1	-1	+1	74	91	10.1
+1	-1	+1	+1	-1	-1	-1	55	99	10.3
-1	+1	-1	+1	+1	-1	-1	117	123	10.1
-1	-1	+1	-1	+1	+1	-1	51	77	10.3
+1	+1	+1	+1	+1	+1	+1	76	81	10.2

**Table 1.** First seven columns gives the P&B design matrix for  $N = 4, 5, 6,$  and  $7$ . Last three columns contain the execution time Q1, Q2, and Q3 used in the illustration example. Upper eight rows contain base matrix, and lower eight rows are needed if *foldover* is used.

Table 1. Each row of the matrix corresponds to one experiment. Each cell in the matrix corresponds to the value used for a parameter in an experiment. Entry in the matrix is either “+1” or “-1”. “+1” corresponds to a value slightly higher than the normal range of values for that parameter and “-1” corresponds to a value slightly lower than the normal range of that parameter. “+1” and “-1” values are not restricted to only numeric values. For example, for buffer page replacement algorithm, the “-1” value can be “RANDOM” and “+1” value can be “CLOCK”. Experiments are conducted by setting up the values according to the design matrix, and responses are recorded. The net effect of each parameter is estimated by multiplying the response value with the corresponding “+1” or “-1” for each row and summing the values across all rows. The absolute value of net effect is used to determine the relative importance of that parameter.

## 4 P&B Design for DBMS

SARD uses P&B design to estimate the effects of configuration parameters on DBMS performance. It has three major steps: 1) for each query, estimation of the effects of each configuration parameter for each query; 2) ranking parameters based on relative magnitude of effects; and 3) combining the rankings of all individual queries to determine the final ranking of the parameters for entire workload. Sections 4.1, 4.2, and 4.3 discuss these three steps in detail, respectively. Section 4.4 discusses some guidelines of applying P&B design in DBMS.

### 4.1 Effect Estimation

At first, a *P&B design matrix* is constructed, which gives the specification of values used for each parameter in each experiment. The dimension of design matrix depends on number of configuration parameters,  $N$ . The design matrix has  $X$  rows and  $X - 1$  columns, where  $X$  is the next multiple of 4 greater than  $N$ . For example, if  $N = 5$ ,

then  $X = 8$ ; if  $N = 8$ , then  $X = 12$ . In the design of optimal multifactorial experiments work, Plackett and Burman recommended the optimal starting setting for  $X = 8, 12, 16, \dots, 96, 100$  [28]. Based on their recommendations, the first row of the *P&B design matrix* is selected from [28], for the corresponding value of  $X$ . Rest of the  $(X - 1)$  rows of the *P&B design matrix* are constructed by right cyclic shifting of the immediate preceding row. All entries of the  $X$ -th row of the P&B design matrix are set to “-1”s.

An improvement that increase the accuracy of the base P&B design is the P&B design with *foldover* [26]. However, if *foldover* is used, there will be  $X$  extra rows in the base *P&B design matrix*. These additional rows are constructed by reversing the sign of the entries of top  $X$  rows.  $(X + i)$ -th row is formed by reversing the sign of  $i$ -th row. If  $N < (X - 1)$ , means that number of columns in the P&B design matrix is more than the number of configuration parameters. In this case, the additional  $(X - N - 1)$  last columns of the P&B matrix are considered as dummies, are simply ignored. For each query of the workload,  $i$ -th experiment is conducted by setting the parameters values of the configuration parameters according to the  $i$ -th row of the P&B design matrix, and execution time is recorded. The effect of the each configuration parameter is calculated by multiplying the corresponding “+1” or “-1” of that parameter in the  $i$ -th row of the P&B matrix with the recorded query execution time, and summing up the products across all rows of the design matrix.

SARD can also determine the sensitivity of a query to parameter tuning. For each query, SARD calculates the *standard deviation* of the net effects for the all configuration parameters. If standard deviation of the effects is very low, that means the query performance will not be affected by the change in configuration parameter settings.

For illustration, consider that seven configuration parameters  $A, B, C, D, E, F, G$  need to be ranked. The workload consists of only three queries: Q1, Q2, and Q3. The first seven columns in Table 1 corresponds to the design matrix for seven parameters and the last three columns correspond to the execution times of the query Q1, Q2, and Q3 for each experiment. As  $N = 7$ , the value of  $X$  will be 8, as the next multiple of four greater than four is eight. For the base case, we need to conduct eight experiments. If *foldover* is used, we need to conduct 16 experiments. In this paper, we assume that *foldover* is used. The first row of the *P&B design matrix* in Table 1, is copied from [28] according the value of  $X = 8$ . For Q1, 16 experiments are conducted and execution is recorded, as shown in the eighth column of Table 1. The net effect of the first parameter  $A$  for Q1 is calculated by multiplying the entries in the first column with entries in the eighth column and summing up across all 16 rows. For Q1, the net effect of the parameter  $A$  is estimated as:

$$Effect_A = (+1 * 34) + (-1 * 19) + \dots + (-1 * 51) + (+1 * 76) = -109.$$

Similarly, the net effect of the second parameter  $B$  for

	A	B	C	D	E	F	G	stdev
Q1	-109	79	-167	193	21	-25	177	136.4
Q2	-61	161	9	143	39	-185	109	123.3
Q3	0.40	-0.80	0.00	0.40	0.40	0.00	0.40	0.44

**Table 2.** The P&B effects for Q1, Q2, and Q3.

	A	B	C	D	E	F	G
Q1	-0.6	0.4	-0.9	1	0.1	-0.1	0.9
Q2	-0.3	0.9	0.0	0.8	0.2	-1.0	0.6

**Table 3.** The P&B normalized effects with respect to the maximum effect for Q1 and Q2.

Q1 is calculated by multiplying the entries in the second column with the entries in eighth column and summing across all 16 rows; The net effect of  $A$  for Q2 is calculated by multiplying the entries in the first column with entries in the ninth column and summing across all 16 rows, and so on. The effects of all seven parameters for Q1, Q2, and Q3 are shown in Table 2. The last column in Table 2 gives the standard deviation of net effects of the parameters for Q1, Q2, and Q3 are 136.4, 123.3, and 0.44, respectively. As the standard deviation of Q3 is very small, it will not be affected by configuration parameters tuning.

## 4.2 Parameter Ranking for a Query

Once the effects of all configuration parameters and the sensitivity of a query is determined, the next step is to rank parameters. If it is insensitive to parameter changes, the ranking can safely be ignored; the performance of an insensitive query is unaffected by parameter values changes. Ranking can be done in various ways. Here we discuss two possible alternatives. A relative comparison between them is discussed in Section 6.

**Sorting Effects:** The effects of the parameters are sorted in descending order of their magnitude. Rank is assigned in the sorted order. Using the data from Table 2 for Q1 the ranking of the parameters A, B, C, D, E, F, and G will be 4, 5, 3, 1, 7, 6, and 2 respectively. Similarly, for Q2 the corresponding rankings are 5, 2, 7, 3, 6, 1, and 4. All the ranks form a *rank vector*. The problem with this simple approach is that if some effects are very close to each other, intuitively it means the corresponds parameters effects are all the same. But the sorting method assigned them different ranking order. For example, there are 4 parameters P, Q, R, and S and their effects are 1500, 50.6, 51.4, and 3000, respectively. The ranking due to sorting method will be 2, 4, 3, and 1. But the effects of Q and R are very close. Since the effects are similar they should be assigned the same rank.

**Rounding Normalized Effects:** To overcome the problem of the simple sorting method, effects are normalized with respect to the maximum effect, rounded to the first decimal point, and sorted in descending order. All the parameters with the same normalized effect are assigned the same rank. For example, in the previous example the normalized

Sorting the effects							
	A	B	C	D	E	F	G
Q1	4	5	3	1	7	6	2
Q2	5	2	7	3	6	1	4

Normalized to max effect							
	A	B	C	D	E	F	G
Q1	4	5	3	1	7	7	3
Q2	5	2	7	3	6	1	4

**Table 4.** Ranking of the configuration parameters for Q1 and Q2. Q3 is not included as it not sensitive to tuning.

Average Effect							
	A	B	C	D	E	F	G
	42.5	60	44	84	15	52.5	71.5
Final Rank	6	3	5	1	7	4	2

Average Rank							
	A	B	C	D	E	F	G
	4.5	3.5	5	2	6.5	3.5	3
Final Rank	5	3	5	1	7	3	2

Average Normalized Effect							
	A	B	C	D	E	F	G
	0.9	1.3	0.9	1.8	0.3	1.1	1.5
Final Rank	6	3	6	1	7	4	2

Euclidean Distance							
	A	B	C	D	E	F	G
	5.0	4.1	6.1	2.0	7.1	5.0	3.2
Final Rank	3	5	6	1	7	3	2

**Table 5.** Overall ranking of the configuration parameters for the workload {Q1, Q2, Q3}. In rank calculation, Q3 is ignored as it is found to be not sensitive to tuning.

effects of P, Q, R, and S are 0.5, 0.0, 0.0, and 1. According to the rounding method the ranks are 2, 4, 4, and 1. According to this method, the ranking vectors for Q1 and Q2 in Table 3 are  $\langle 4, 5, 3, 1, 7, 7, 3 \rangle$  and  $\langle 5, 2, 7, 3, 6, 1, 4 \rangle$ .

Table 5 summarizes the ranks of the configuration parameters, calculated using the two alternative described above, for the query Q1 and Q2.

### 4.3 Parameter Ranking for the Workload

After determining the ranking of the parameters for each individual query, next step is to estimate the ranking of parameters for the entire workload. The queries which are insensitive to parameter tuning, are not included in workload ranking calculation. In this section, we will discuss several alternatives for determining ranking for the workload. A relative comparison among the alternatives is discussed in Section 6.

**Average Effect:** Average the P&B effects of the parameters across all queries, and rank them in the descending order. For the Q1 and Q2 in Table 2 using this approach, the total sum of the effects of parameters A, B, C, D, E, F, and G are found to be 85, 120, 88, 168, 30, 105, and 143, respectively. The final ranking for the workload is  $\langle 6, 3, 5, 1, 7, 4, 2 \rangle$ . If a query has longer execution time, the effects of the parameters for that query are also large compared to a query which has relatively smaller execution time. So the topmost effects of the longest running query can dominate the effects of other queries, and distort the ranking.

**Average Rank:** This method removes the problem of

having one large execution time dominating the effect of the entire workload. Ranks are summed across all queries, averaged and sorted in descending order. The most important parameters will have the lowest cumulative rank. For example, in Table 2, for the entire workload, the rankings of the parameters A, B, C, D, E, F, and G the averaged summation of ranks are 4.5, 3.5, 5, 2, 6.5, 3.5, and 3, respectively. Final ranking vector is  $\langle 5, 3, 5, 1, 7, 3, 2 \rangle$ . This method can suffer from the problem of having a small difference in the magnitude of the effect in individual queries; will have a large impact on overall workload ranking.

**Average Normalized Effect:** By simply averaging the sum of the effects, implies that the all queries have equal importance. But differences in execution time imply different importance of each query for the workload. To take this condition in account, we can normalize the effects with respect to the total sum of effects for each query. The average of summation of normalized effects across queries can be sorted in descending order to determine final ranking of workload. For example, for the normalized effects in Table 3, if one considers the ranking of query using averaging ranking, the average ranking of parameters A, B, C, D, E, F, and G will be 0.9, 1.3, 0.9, 1.8, 0.3, 1.1, and 1.5, respectively. Final ranking will be  $\langle 6, 3, 6, 1, 7, 4, 2 \rangle$ .

**Euclidean Distance:** If a parameter is the most important for a query, the ranking of the corresponding parameter will be ranked one for that query. If there are  $m$  queries, a parameter is the most important for all the  $m$  queries the rank vector of that parameter across the queries will be  $\langle 1_{q_1}, 1_{q_2}, \dots, 1_{q_{m-1}}, 1_{q_m} \rangle$ . We define it as *ideal unit vector*. Now for each parameter, we can form a vector of rank  $\langle rank_{q_1}, rank_{q_2}, \dots, rank_{q_{m-1}}, rank_{q_m} \rangle$  across  $m$  queries and calculate the Euclidean distance with respect of the *ideal unit vector*. Rank is assigned increasing order of Euclidean distance. A potential pitfall of this method is that one poor ranking will mask the large impact upon the ranking. One of the potential solutions is to flip the *ideal unit vector* as  $\langle m_{q_1}, m_{q_2}, \dots, m_{q_{m-1}}, m_{q_m} \rangle$ , treat rank  $m$  being the most sensitive parameter. For the Q1 and Q2, the ideal unit vector will be  $\langle 7, 7, \dots, 7, 7 \rangle$ . The new rank (highest value is most sensitive) from the old rank (lowest rank is most sensitive) is calculated using  $(m + 1 - Rank_{old})$ . The Euclidean distance for the parameter A using the new ranking is  $\sqrt{(4 - 7)^2 + (3 - 7)^2} = 5$ . Similarly, the Euclidean distance of parameters B, C, D, E, F, and G with respect to unit vector are 4.1, 6.1, 2, 7.1, 5.3, and 3.2, respectively. Final ranking for the workload is  $\langle 3, 5, 6, 1, 7, 3, 2 \rangle$ .

Table 5 summarizes the overall ranking of the configuration parameters, calculated using the four different methods described above, for the illustration workload .

### 4.4 Discussion

To apply SARD, a common dilemma is whether it is better to use a too large range or a too small range. The prob-

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

**Table 6.** Machine Description

lem of using a too large range is that it will artificially inflate the effect of that parameter. The problem with using a too small range is that the parameter will appear to have no net effect. In general, using a too large range is better than that of a too small range. The intuition behind this is that at least in the former case, the parameter will appear to have an effect.

Again, it does not matter whether “+1” value causes lowest performance or “-1” value causes highest performance. Because each parameter is set to ‘+1’ and “-1” for half of the configurations, and the final effect is calculated by taking the absolute value.

As SARD considers two extreme values of a parameter to estimate the net effect of that parameter on DBMS performance, problem arises when the performance (e.g. execution time) is not a monotonic function of a parameter value. For example, in PostgreSQL, with the increase of the size of `shared_buffers` the performance increases till a certain value of `shared_buffers`, once threshold is exceeded, with the increase of `shared_buffers` size the performance decreases. To solve this problem, using intuitions, DBAs split the original range of the parameter into subranges where it acts as monotonic and then treat each of these subranges as a separate parameter.

## 5 Experimental Setup

In this section, we will give an overview of the DBMS parameters, workload, and the machine setup used for collecting experimental data. The description of the machine used for conducting experiments is given in Table 6.

### 5.1 Benchmark Used

For demonstration, we have used TPC-H benchmark [3]. TPC-H database is populated by data generation program *dbgen* with *scaling factor* (SF) of one (data size is 1GB). TPC-H consists of 22 read only and two update queries. We have considered only read only queries. Queries are generated by *qgen* program supplied with TPC-H benchmark. For collecting data, queries are modified by adding EXPLAIN ANALYZE. Queries are executed one at a time. We form two workloads, {Q1, Q8, Q9, Q13, Q16} and {Q2, Q3, Q6, Q7, Q21}, to show that ranking varies with workload characteristics changes. The workloads are formed randomly. Finally, we will find a ranking for the entire TPC-H benchmark read only queries.

### 5.2 Parameters Considered

For demonstration, we have used the PostgreSQL8.2 DBMS PostgreSQL has approximately 100 configuration parameters [2]. As our queries are read only, many of

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

**Table 7.** PostgreSQL configuration parameters and their P&B values used.

the parameters are not relevant. We have considered only those parameters which seem to be relevant to read only queries. We have also deliberately selected parameters `checkpoint_timeout` and `fsync` which do not have any effect on read only queries. Intuitively the relative impact of this type of parameters should be low or zero for read only queries. Including them, will help us to validate our methodology. The high and low P&B values for the parameters used in this demonstration are given in Table 7. The values are selected according to the recommendations made in PostgreSQL documentation [2]. For demonstration, we choose the high and low values for each parameter in a range such that it will act as monotonic.

## 6 Results

In this section, first we will demonstrate that ranking varies depending on query workload. Next, we will show that SARD can correctly identify performance critical configuration parameters. Finally, we will classify TPC-H queries into two groups based on their sensitivity to configuration parameter tuning.

### 6.1 Workload Ranking

To demonstrate ranking is not static, rather varies and depends on query workload, we rank the configuration parameters for two workloads consisting of {Q1, Q8, Q9, Q13, Q16} and {Q2, Q3, Q6, Q7, Q21}. Table 8 and Table 10 show the estimated P&B effects and ranking of the configuration parameters for the queries of workload one and two, respectively. Ranking is calculated using both *sorting* methodology and *rounding the normalized effect* methodology. We will refer *rounding the normalized effect* methodology as *rounding* methodology in the rest of the discussion.

Compared to the *sorting* methodology, the *rounding* methodology clusters the parameters which have the similar P&B effects. Although effects can be different in magnitude from each other, but compared to the maximum effect, they may have marginal or no impact on system perfor-

Parameter	P&B Effects					Ranking by Sorting					Ranking by Rounding Normalized Effect				
	Q1	Q8	Q9	Q13	Q16	Q1	Q8	Q9	Q13	Q16	Q1	Q8	Q9	Q13	Q16
Checkpoint_timeout	63139	8910	602068	4130	1590	9	14	5	7	10	15	15	5	15	15
cpu_index_tuple_cost	10651	42106	149179	2460	1697	13	9	12	12	8	15	12	13	15	15
cpu_operator_cost	81071	46111	349389	480	1714	5	7	8	15	7	15	12	11	15	15
cpu_tuple_cost	58019	74791	48262	5808	1836	11	5	15	4	5	15	6	15	15	15
deadlock_timeout	11923	64500	54715	5214	1546	12	6	14	5	11	15	6	15	15	15
effective_cache_size	75846	11954	304263	238544	979	6	13	10	1	15	15	15	11	1	15
fsync	59809	4534	274194	3424	1329	10	15	11	10	12	15	15	11	15	15
geqo	3858	82480	529011	853	1679	15	3	7	14	9	15	6	7	15	15
maintenance_work_mem	75774	127031	780976	1351	1048	7	2	2	13	14	15	2	3	15	15
max_connections	97626	77876	628112	3678	1792	4	4	4	8	6	15	6	5	15	15
random_page_cost	8693	43699	1694532	3447	2145	14	8	1	9	3	15	12	1	15	15
shared_buffers	162386	34787	770430	132957	5997	2	10	3	2	2	15	12	3	2	2
stats_start_collector	71420	25181	328104	4278	2122	8	12	9	6	4	15	12	11	15	15
temp_buffers	101473	31217	571133	2713	1238	3	11	6	11	13	15	12	7	15	15
work_mem	5523703	359544	142035	24839	63760	1	1	13	3	1	1	1	13	3	1

**Table 8.** P&B effects of the configuration parameters for the queries of workload one.

Rank	Average Rank	Averaging Effect	Average Normalized Effect	Euclidean Distance
1	work_mem	work_mem	work_mem	work_mem
2	shared_buffers	random_page_cost	effective_cache_size	shared_buffers
3	maintenance_work_mem	shared_buffers	shared_buffers	maintenance_work_mem
4	max_connections	maintenance_work_mem	random_page_cost	effective_cache_size
5	effective_cache_size	max_connections	maintenance_work_mem	random_page_cost
6	geqo	temp_buffers	max_connections	max_connections
7	random_page_cost	Checkpoint_timeout	geqo	geqo
8	Checkpoint_timeout	effective_cache_size	temp_buffers	Checkpoint_timeout
9	temp_buffers	geqo	Checkpoint_timeout	cpu_tuple_cost
10	cpu_tuple_cost	cpu_operator_cost	cpu_operator_cost	deadlock_timeout
11	deadlock_timeout	stats_start_collector	stats_start_collector	temp_buffers
12	cpu_operator_cost	fsync	cpu_tuple_cost	cpu_operator_cost
13	stats_start_collector	cpu_index_tuple_cost	deadlock_timeout	stats_start_collector
14	cpu_index_tuple_cost	cpu_tuple_cost	cpu_index_tuple_cost	fsync
15	fsync	deadlock_timeout	fsync	cpu_index_tuple_cost

**Table 9.** The final ranking of the configuration parameters for workload one.

mance. *Rounding* methodology gives us more fine-grained knowledge of the importance of the parameters for tuning an individual query. For example, in Table 8, for Q1, *sorting* methodology gives 15 different rankings for the configuration parameters. In contrast, *rounding* methodology gives us the information that only `work_mem` is important for tuning Q1, other parameters have a very small impact compared to the topmost parameter.

If we consider *rounding* methodology, the results from Table 8 indicates that for Q1, `work_mem` is the most important parameter; for Q8, `work_mem` and `maintenance_work_mem` are the most important parameters; for Q9 `random_page_cost`, `shared_buffers`, `maintenance_work_mem` are the most important parameters; for Q13, `effective_cache_size`, `shared_buffers` are most important parameters; and for Q16, `work_mem` and `shared_buffers` are important parameters. `work_mem` is ranked first in Q1, Q8, and Q9. `maintenance_work_mem` is ranked second and third respectively for Q8 and Q9. `shared_buffers` is ranked third, second, and second respectively in Q9, Q13, and Q16. `effective_cache_size` is ranked one in Q3. If we assume that all queries have same importance in the workload, intuitively it appears that `work_mem` is the most important configuration to optimize, as it appears as ranked first for three queries out of five, for the workload one. `shared_buffers` also looks very promising for tuning.

The results in Table 9 estimate the overall ranking of the all configuration parameters for workload one. Ranks are calculated using four different alternatives described in Sec-

tion 4.3. Although different alternatives give different ranking, the most important parameters are consistent across the different methods. It is also interesting to note that overall ranking of the configuration parameters for the workload, are different from their rankings for the individual queries of the workload.

Considering *rounding* methodology, the results in Table 8, gives an indication that `work_mem` is the most important configuration to optimize, as it is ranked first for three out of five queries of workload one. `shared_buffers` also looks very promising for tuning as it is also ranked second for three out of five queries. For all alternatives, `work_mem` and `shared_mem` are always in the among three topmost important parameters. `random_page_cost` and `effective_cache_size` are also appear to be important for all the alternatives in Table 9 except alternative one and two, respectively. With the exception of the strategy two, the ranking closely matches with each other. `maintenance_work_mem` also appears in the topmost five parameters in all alternatives in Table 9. From the results, it appears that all the alternatives give consistent overall ranking.

Another interesting observation is that `fsync` and `checkpoint_timeout` never appear most important for the individual queries as well as the entire query workload. As the queries we considered here are read only, intuitively these two parameters should have marginal impact to improve the performance of the overall query workload. Experimental results match with intuition. The low ranking of these two parameters validates that SARD is work-

Parameter	P&B Effects					Ranking by Sorting					Ranking by Rounding Normalized Effect				
	Q2	Q3	Q6	Q7	Q21	Q2	Q3	Q6	Q7	Q21	Q2	Q3	Q6	Q7	Q21
Checkpoint_timeout	28711	11325	0.105	2269	207883	12	15	NA	3	9	2	15	NA	2	1
deadlock_timeout	5582	16801	0.125	1968	100185	15	13	NA	4	13	15	15	NA	15	6
cpu_operator_cost	130508	57606	0.076	158	15003	3	2	NA	14	15	15	7	NA	15	6
stats_start_collector	28909	22561	0.071	1200	179365	11	10	NA	8	10	15	15	NA	15	6
cpu_tuple_cost	48172	31252	0.037	718	398057	7	7	NA	13	6	1	7	NA	15	6
effective_cache_size	6874535	43373	0.034	1326	627016	1	5	NA	6	3	15	15	NA	15	6
fsync	13556	20707	0.053	1027	31385	13	11	NA	10	14	15	1	NA	1	11
geqo	30488	56499	0.118	1247	141108	10	3	NA	7	11	15	15	NA	15	11
maintenance_work_mem	74429	26012	0.084	135	413680	5	8	NA	15	5	15	3	NA	15	11
temp_buffers	42060	45193	0.064	987	240543	9	4	NA	11	8	15	15	NA	15	11
shared_buffers	2459527	25049	0.031	36200	2632182	2	9	NA	2	1	15	7	NA	15	11
random_page_cost	46948	17519	0.114	855	490531	8	12	NA	12	4	15	15	NA	15	15
max_connections	68939	34326	0.103	1163	117188	6	6	NA	9	12	15	7	NA	15	15
cpu_index_tuple_cost	103431	11380	0.057	1555	657371	4	14	NA	5	2	15	3	NA	15	15
work_mem	13295	183039	0.017	79932	310306	14	1	NA	1	7	15	15	NA	15	15

**Table 10.** P&B effects of the configuration parameters for the queries of workload two.

Rank	Average Rank	Averaging Effect	Average Normalized Effect	Euclidean Distance
1	shared_buffers	effective_cache_size	work_mem	shared_buffers
2	work_mem	shared_buffers	shared_buffers	work_mem
3	effective_cache_size	cpu_index_tuple_cost	effective_cache_size	effective_cache_size
4	cpu_tuple_cost	work_mem	geqo	geqo
5	geqo	random_page_cost	temp_buffers	cpu_tuple_cost
6	temp_buffers	maintenance_work_mem	cpu_index_tuple_cost	cpu_operator_cost
7	cpu_operator_cost	cpu_tuple_cost	cpu_operator_cost	cpu_index_tuple_cost
8	maintenance_work_mem	temp_buffers	cpu_tuple_cost	maintenance_work_mem
9	random_page_cost	Checkpoint_timeout	maintenance_work_mem	random_page_cost
10	cpu_index_tuple_cost	stats_start_collector	random_page_cost	temp_buffers
11	max_connections	geqo	max_connections	max_connections
12	Checkpoint_timeout	max_connections	stats_start_collector	Checkpoint_timeout
13	stats_start_collector	cpu_operator_cost	Checkpoint_timeout	stats_start_collector
14	deadlock_timeout	deadlock_timeout	deadlock_timeout	deadlock_timeout
15	fsync	fsync	fsync	fsync

**Table 11.** The ranking of the configuration parameters for workload two.

ing correctly. Although for the individual queries sometimes `fsync` and `checkpoint_timeout` appears relatively high, but ranking by *rounding* methodology indicates that these parameters have very small impact.

Table 10 gives the ranking configuration parameters for the individual queries of workload two. There is an interesting result to notice; the ranking for Query Q6 does appear as not applicable (N/A). Considering the net effects of all parameters, it is clear that the differences among the effects for Q6 are very small, they are virtually all equivalent. Due to this for Q6, ranking of the configuration parameters will be misleading. It is insensitive to parameters tuning ranking does not make any real sense to improve performance.

Table 11 gives the overall ranking of the configuration parameters for the workload two using different alternatives described in Section 4.3. As Q6 is insensitive to parameter tuning, it is excluded when the results in Table 11 are calculated. From the results in Table 9 and Table 11, it is clear that the ranking of configuration parameters vary with the query workload.

## 6.2 Ranking Effectiveness

To demonstrate that ranking gives correct order of the configuration parameters based on their relative impact on the DBMS performance, we find the tuned values for first seven important parameters of workload one using *averaged effect* methodology. The default values and tuned values are given in Table 12. We find that the tuned values of the last four parameters are same as default values shipped with PostgreSQL. As a result, if we tune only last four pa-

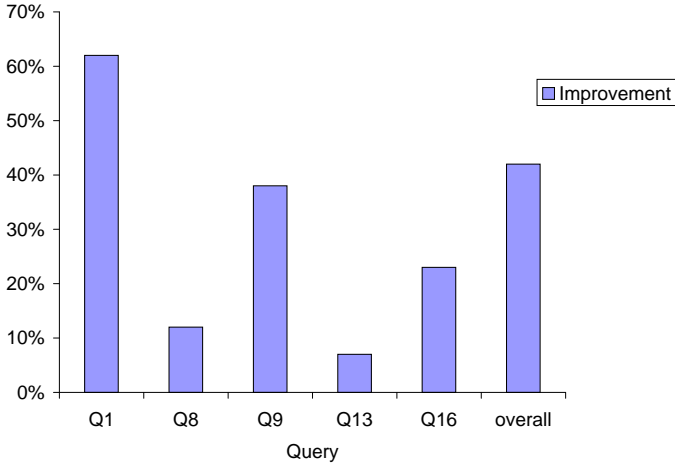
Parameter	Default	Tuned
work_mem	1MB	128MB
shared_buffers	32MB	96 MB
random_page_cost	4	2.5
maintenance_work_mem	16MB	default
temp_buffers	8MB	default
max_connections	100	default
effective_cache_size	128MB	default

**Table 12.** Default PostgreSQL values and tuned values for seven configuration parameters used for tuning the query workload one.

rameters we do not get any performance improvement. On the other hand, if we tune first three values, according to Figure 1 overall performance of the query workload one improves by 42% and individually query performance of Q1, Q8, Q9, Q13, and Q16 improves by 62%, 12%, 38%, 7%, and 23%, respectively. This clearly shows that SARD's estimated ranking gives correct order of configuration parameter according to their impact on system performance.

Again, to show the robustness of SARD, we also estimate the ranking by adding a new query and changing relative weights of the queries in the workload one consisting of {Q1, Q8, Q9, Q13, Q16}.

*Adding New Query* : Query Q15 and Q19 are added to workload one. We find that Q15 is insensitive to parameter tuning. So overall ranking for the workload one remains unchanged. For Q19 the four topmost important configuration parameters are `shared_buffers`, `effective_cache_size`, `cpu_operator_cost`, and



**Figure 1.** Workload one improvement due to tuning compared to the default settings.

Tuning	Query
Sensitive	Q1, Q2, Q3, Q5, Q7, Q8, Q9, Q11, Q13, Q16, Q17, Q18, Q19, Q20, Q21, Q22
Insensitive	Q4, Q6, Q10, Q12, Q14, Q15

**Table 13.** Classification of the TPC-H queries.

`cpu_tuple_cost`, in order. We find that even after adding Q19, the ranking remains unchanged. This is not unusual, as `work_mem` is ranked one in three queries out of six on workload one, so intuitive it should be ranked one for the overall workload. It is interesting to note that the ranking of the parameters `cpu_operator_cost` and `cpu_tuple_cost` is low in the overall ranking of the workload. In order to improve Q19’s performance these two parameters need to be tuned. Our results support this claim. We find that using the tuned value of Table 12, Q19’s performance improved by 3%, but overall performance of the new workload improved by 40% compared to the default setting values.

*Changing Relative Weight:* So far we assume that all queries are equally important. We assign new weight of 0.4, 0.2, 0.2, 0.1, and 0.1, respectively to Q1, Q8, Q9, Q13, and Q16. The ranking is assigned by the relative importance of the queries. We find that ranking using new *weighted sum of effects* swapped ranking between `checkpoint_timeout` and `effective_cache_size`, `cpu_operator_cost` and `cpu_index_tuple_cost` compared to the earlier ranking using *sum of effects*. After tuning, we find that the overall performance of workload one improved by 48% compared to the default setting, and 14.29% compared to the unweighted tuned queries performance.

### 6.3 Classification of the TPC-H Queries

There are queries for which performance does not improve at all with the configuration parameter tuning. We de-

fine them as *queries insensitive to tuning*. Our experiments results shows that out of 22 read only queries of the TPC-H benchmark, query Q4, Q6, Q10, Q12, Q14, and Q15 are insensitive to parameter tuning. The execution time does not vary at all for these insensitive queries. As execution time is almost same across all P&B experiments, the net effects of configuration parameters for these queries are virtually equivalent. Our experimental results show that these queries are not completely insensitive to parameter tuning, but the relative performance gain from parameter tuning is very less compared to the queries which are sensitive to tuning. Table 13 gives the classification of the TPC-H queries based on the sensitivity to the parameter tuning.

We also estimate the overall ranking of all configuration parameters for the workload consisting of the read only queries of the TPC-H benchmark, ignoring the tuning insensitive queries. The rankings are given in Table 14.

## 7 Conclusion

SARD provides a methodology for ranking the configuration parameters based on their relative impact on DBMS performance. SARD is a generic methodology and can be applied to the non-database systems as well. Our experimental results indicate that the ranking of configuration parameters are highly dependent upon the query workload. As a result, using generic rules of thumb for tuning DBMS based on generic workloads may waste DBAs time and effort. A ranking based on statistical data will be a great aid for the DBAs to direct the tuning efforts. SARD can greatly serve this purpose. Experimental results also show that SARD can identify critical configuration parameters, and tuning top ranked parameters can greatly enhance DBMS performance. From our experiences in this work, we have come to the following two conclusions. *First, when tuning individual queries, DBAs can use the ranking by rounding normalized effects method. Second, since the ranking of parameters for individual queries and the entire workload are different, DBAs must examine both to have a better idea of what parameters are important to consider for improving DBMS performance.*

## References

- [1] Oracle Tuning Pack. [http://www.oracle.com/technology/products/manageability/database/pdf/ds/tuning\\_pack\\_ds\\_10gr1.pdf](http://www.oracle.com/technology/products/manageability/database/pdf/ds/tuning_pack_ds_10gr1.pdf).
- [2] PostgreSQL DBMS Documentation. <http://www.postgresql.org/>.
- [3] Transaction Processing Council. <http://www.tpc.org/>.
- [4] S. Agrawal, S. Chaudhuri, L. Kollar, A. Marathe, V. Narasayya, and M. Syamala. Database Tuning Advisor for Microsoft SQL Server 2005. In *Proceedings of VLDB*, pages 1110–1121, 2004.
- [5] S. Agrawal, S. Chaudhuri, and V. Narasayya. Automated selection of materialized views and indexes in sql databases. In *Proceedings of VLDB*, pages 496–505, 2000.

Rank	Average Rank	Averaging Effect	Average Normalized Effect	Euclidean Distance
1	shared_buffers	effective_cache_size	shared_buffers	shared_buffers
2	cpu_operator_cost	work_mem	work_mem	work_mem
3	cpu_tuple_cost	shared_buffers	effective_cache_size	effective_cache_size
4	effective_cache_size	cpu_operator_cost	cpu_operator_cost	cpu_operator_cost
5	maintenance_work_mem	maintenance_work_mem	cpu_tuple_cost	random_page_cost
6	random_page_cost	max_connections	random_page_cost	maintenance_work_mem
7	work_mem	random_page_cost	maintenance_work_mem	cpu_tuple_cost
8	Checkpoint_timeout	temp_buffers	max_connections	max_connections
9	gsqo	cpu_index_tuple_cost	gsqo	fsync
10	max_connections	Checkpoint_timeout	fsync	temp_buffers
11	fsync	gsqo	Checkpoint_timeout	gsqo
12	temp_buffers	cpu_tuple_cost	temp_buffers	Checkpoint_timeout
13	cpu_index_tuple_cost	stats_start_collector	cpu_index_tuple_cost	cpu_index_tuple_cost
14	stats_start_collector	fsync	deadlock_timeout	deadlock_timeout
15	deadlock_timeout	deadlock_timeout	stats_start_collector	stats_start_collector

**Table 14.** Final ranking of the configuration parameters for the TPC-H workload.

- [6] S. Agrawal, V. Narasayya, and B. Yang. Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design. In *Proceeding of SIGMOD*, pages 359–370, 2004.
- [7] N. Bruno and S. Chaudhuri. Automatic Physical Database Tuning: A Relaxation-based Approach. In *Proceeding of SIGMOD*, pages 227–238, 2005.
- [8] N. Bruno and S. Chaudhuri. To Tune or not to Tune?: A Lightweight Physical Design Alerter. In *Proceeding of VLDB*, pages 499–510, 2006.
- [9] S. Chaudhuri and V. Narasayya. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In *Proceeding of VLDB*, pages 146–155, 1997.
- [10] S. Chaudhuri and V. Narasayya. AutoAdmin What-If Index Analysis Utility. In *Proceeding of SIGMOD*, pages 367–378, 1998.
- [11] S. Chaudhuri and G. Weikum. Rethinking Database Architecture: Towards a Self-tuning RISC-style Database System. In *Proceedings of VLDB*, pages 1–10, 2000.
- [12] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait, and M. Ziauddin. Automatic SQL Tuning in Oracle 10g. In *Proceedings of VLDB*, pages 1098–1109, 2004.
- [13] B. Dageville and K. Dias. Oracle’s Self-Tuning Architecture and Solutions. *IEEE Data Engineering Bulletin*, 29(3):24–31, 2006.
- [14] K. Dias, M. Ramacher, U. Shaft, V. Venkataramamani, and G. Wood. Automatic Performance Diagnosis and Tuning in Oracle. In *Proceedings of CIDR*, pages 1110–1121, 2005.
- [15] C. Garry. Who’s Afraid of Self-Managing Databases? <http://www.eweek.com/article2/0,1895,1833662,00.asp>, June 30, 2005.
- [16] G. Group. The Total Cost of Ownership: The Impact of System Management Tools. 1996.
- [17] H. Group. Achieving Faster Time-to-Benefit and Reduced TCO with Oracle Certified Configurations. March, 2002.
- [18] I. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. In *Proceeding of SIGMOD*, pages 647–658, 2004.
- [19] A. Joshi, J. Yi, R. Bell, L. Eeckhout, L. John, and L. David. Evaluating the Efficacy Statistical Simulation for Design Space Exploration. In *Proceedings of ISPASS*, 2006.
- [20] E. Kwan, S. Lightstone, A. Storm, and L. Wu. Automatic Configuration for IBM DB2 Universal Database. In *IBM Performance Technical Report*, January 2002.
- [21] S. Lightstone and B. Bhattacharjee. Automated Design of Multidimensional Clustering Tables for Relational Databases. In *Proceeding of VLDB*, pages 1170–1182, 2004.
- [22] 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. *IEEE Data Engineering Bulletin*, 29(3):16–23, 2006.
- [23] D. J. Lilja. *Measuring Computer Performance A practitioner’s Guide*. Cambridge University Press, 2000.
- [24] G. Lohman and S. Lightstone. SMART: Making DB2 (More) Autonomic. In *Proceeding of VLDB*, pages 877–879, 2002.
- [25] P. Martin, W. Powley, and D. Benoit. Using Reflection to Introduce Self-Tuning Technology Into DBMSs. In *Proceedings of IDEAS*, pages 429–438, 2004.
- [26] D. Montgomery. *Design and Analysis of Experiments*. Wiley, 2001.
- [27] V. Nookala, Y. Chen, D. Lilja, and S. Sapatnekar. Microarchitecture-aware Floorplanning Using a Statistical Design of Experiments Approach. In *Proceedings of DAC*, 2005.
- [28] R. Plackett and J. Burman. The Design of Optimum Multifactorial Experiments. In *Biometrika Vol. 33 No. 4*, pages 305–325, 1946.
- [29] A. Rosenberg. Improving Query Performance in Data Warehouses. <http://www.tdwi.org/Publications/BIJournal/display.aspx?ID=7891>, 2005.
- [30] M. Sopotkamol. Ranking Configuration Parameters in Multi-tiered E-Commerce Sites. *SIGMETRICS Performance Evaluation Review*, 32(3):24–33, 2004.
- [31] M. Sopotkamol and D. A. Menascé. A method for evaluating the impact of software configuration parameters on e-commerce sites. In *WOSP*, pages 53–64, 2005.
- [32] M. Stillger, G. Lohman, V. Markl, and M. Kandil. LEO - DB2’s Learning Optimizer. In *Proceeding of VLDB*, pages 19–28, 2001.
- [33] A. Storm, C. Garcia-Arellano, S. Lightstone, Y. Diao, and M. Surendra. Adaptive self-tuning memory in DB2. In *Proceeding of VLDB*, pages 1081–1092, 2006.
- [34] J. Yi, S. Kodakara, R. Sendag, D. Lilja, and D. Hawkins. Characterizing and Comparing Prevailing Simulation Techniques. In *Proceedings of HPCA*, 2005.
- [35] J. Yi, D. Lilja, and D. Hawkins. A Statistically Rigorous Approach for Improving Simulation Techniques. In *Proceedings of HPCA*, 2003.
- [36] D. Zilio, J. Rao, S. Lightstone, G. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden. DB2 Design Advisor: Integrated Automated Physical Database Design. In *Proceedings of VLDB*, pages 1087–1097, 2004.