

Exploiting the Impact of Database System Configuration Parameters: A Design of Experiments Approach

Biplob K. Debnath, Mohamed F. Mokbel, and David J. Lilja
University of Minnesota, Twin Cities, USA.
debna004@umn.edu, mokbel@cs.umn.edu, and lilja@ece.umn.edu

Abstract

Tuning database system configuration parameters to proper values according to the expected query workload plays a very important role in determining DBMS performance. However, the number of configuration parameters in a DBMS is very large. Furthermore, typical query workloads have a large number of constituent queries, which makes tuning very time and effort intensive. To reduce tuning time and effort, database administrators rely on their experience and some rules of thumb to select a set of important configuration parameters for tuning. Nonetheless, as a statistically rigorous methodology is not used, time and effort may be wasted by tuning those parameters which may have no or marginal effects on the DBMS performance for the given query workload. Database administrators also use compressed query workloads to reduce tuning time. If not carefully selected, the compressed query workload may fail to include a query which may reveal important performance bottleneck parameters. In this article, we provide a systematic approach to help the database administrators in tuning activities. We achieve our goals through two phases. First, we estimate the effects of the configuration parameters for each workload query. The effects are estimated through a design of experiments-based PLACKETT & BURMAN design methodology where the number of experiments required is linearly proportional to the number of input parameters. Second, we exploit the estimated effects to: 1) rank DBMS configuration parameters for a given query workload based on their impact on the DBMS performance, and 2) select a compressed query workload that preserves the fidelity of the original workload. Experimental results using PostgreSQL and TPC-H query workload show that our methodologies are working correctly.

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 key to success of a business. A typical database management system (DBMS) has hundreds of configuration parameters and the appropriate setting of these parameters plays a critical role in performance. Database administrators (DBAs) are expected to tune the configuration parameters to appropriate values that get the best DBMS performance for the application of interest. The success of tuning depends on many factors including the query workload, relational schemas, as well as the expertise of the DBAs [20]. However, skilled DBAs are becoming increasingly rare and expensive [16]. A recent

Copyright 2008 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

study on information technology versus DBA costs showed that personnel cost is estimated at 47% of the total cost of ownership [13]. As has been recently reported, DBAs spend nearly a quarter of their time on tuning activities [20]. To reduce the total cost of ownership, it is of essence that DBAs focus only on tuning those configuration parameters which have the most impact on system performance for a representative query workload.

Different database configuration parameters have different impact on a DBMS performance. A sound statistical methodology for quantifying the impact of each configuration parameter and the interactions among these parameters on a DBMS performance is to perform a *full factorial design* [17], where all possible combinations of the input values of the configuration parameters are considered. However, the major problem in applying a *full factorial design* in a DBMS is the large number of configuration parameters. For example, PostgreSQL [1] has approximately 100 configuration parameters and all parameters have multiple possible values. Even if each configuration parameter assumes only two values, then, given a query workload of q queries, we have to perform $q * 2^{100}$ experiments at least twice to apply a full factorial design, which is not feasible in terms of time and effort. To avoid this problem, in many cases, DBAs rely on their experience and rules of thumb to select the appropriate values for the configuration parameters. As heuristics based on experience and intuition are often used, time and effort may be wasted to enhance the performance by tuning those parameters that may have no or marginal effects on the overall performance of the given query workload. In general, misdirected tuning efforts increase the total cost of ownership [8, 12, 6, 14].

In this article, we are addressing the following problem: *Given a DBMS, a set of configuration parameters, a range of values for all parameters, and a query workload; estimate the effect of each configuration parameter based on its impact on the DBMS performance for the query workload.* In particular, we propose a methodology based on the PLACKETT & BURMAN (P&B) design [19] to estimate the impact of database system configuration parameters. The main idea is to conduct a linear number of experiments that provide an approximate sampling of the entire search space. In each experiment, the values of the configuration parameters are varied systematically over a specified range of acceptable values. Subsequent analysis of the collected experimental data is used to estimate the effects of the configuration parameters on the DBMS performance for the given query workload. Once we have the estimated effect of each configuration parameter for all workload queries, we can exploit these effects for: (1) ranking the configuration parameters based on the impact on DBMS performance for the entire query workload, and (2) selecting a compressed query workload based on the similarities of performance bottleneck parameters that preserves the fidelity of the original workload.

The rest of this article is organized as follows: Section 2 describes our *design of experiments*-based methodology. The methodology to estimate the effects of the configuration parameters is described in Section 3. Ranking configuration parameter and selecting a compressed workload are explained in Section 4. Experimental results are described in Section 5. Section 6 highlights related work. Finally, Section 7 concludes the article.

2 Design of Experiments Based Methodology

The simplest design strategy to quantify the impact of all factors and interactions is to apply a *full factorial design*, for example, ANOVA [17], in which system response is measured for all possible input combinations. However, a *full factorial design* requires an exponential number of experiments. To reduce the number of experiments, we make two assumptions. First, the provoked response, such as the total execution time, is a monotonic function of the input parameter values. This indicates that for each configuration parameter, we can consider only two 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. Second, according to the *sparsity of effects principle*, system response is largely dominated by a few main factors and low-order interactions. As a consequence, we can safely ignore the effects of higher order interactions. Based on these assumptions, we use a *two-level factorial design* methodology named PLACKETT & BURMAN (P&B) design [19], which requires only linear number of experiments.

	P&B Design Matrix							Execution Time		
	P_1	P_2	P_3	P_4	P_5	P_6	P_7	Q1	Q2	Q3
Exp_1	+1	+1	+1	-1	+1	-1	-1	34	110	10.2
Exp_2	-1	+1	+1	+1	-1	+1	-1	19	72	10.1
Exp_3	-1	-1	+1	+1	+1	-1	+1	111	89	10.3
Exp_4	+1	-1	-1	+1	+1	+1	-1	37	41	10.3
Exp_5	-1	+1	-1	-1	+1	+1	+1	61	96	10.2
Exp_6	+1	-1	+1	-1	-1	+1	+1	29	57	10.2
Exp_7	+1	+1	-1	+1	-1	-1	+1	79	131	10.3
Exp_8	-1	-1	-1	-1	-1	-1	-1	19	47	10.1
Exp_9	-1	-1	-1	+1	-1	+1	+1	135	107	10.3
Exp_{10}	+1	-1	-1	-1	+1	-1	+1	56	74	10.3
Exp_{11}	+1	+1	-1	-1	-1	+1	-1	112	48	10.1
Exp_{12}	-1	+1	+1	-1	-1	-1	+1	74	91	10.1
Exp_{13}	+1	-1	+1	+1	-1	-1	-1	55	99	10.3
Exp_{14}	-1	+1	-1	+1	+1	-1	-1	117	123	10.1
Exp_{15}	-1	-1	+1	-1	+1	+1	-1	51	77	10.3
Exp_{16}	+1	+1	+1	+1	+1	+1	+1	76	81	10.2

Table 1: The P&B design matrix with foldover for $N = 7$. Execution time of queries Q1-Q3 are in the last three columns.

For each experiment of the P&B design, the value of each parameter is given by a prescribed *P&B design matrix*. Table 1 gives an example of the *design matrix* for the seven parameters $P_1, P_2, P_3, P_4, P_5, P_6,$ and P_7 depicted by the columns 2-8. The Exp_i indicates the values of the configuration parameters that will be used in the i -th experiment. An entry in the parameter columns of the design matrix is either “+1” or “-1”, that corresponds to a value slightly higher or lower than the normal range of values for the corresponding parameter, respectively. The “+1” and “-1” values are not restricted to only numeric values. For example, for the buffer page replacement algorithm, the “-1” value can be “RANDOM” and “+1” value can be “CLOCK”. The *P&B design matrix* is constructed by cyclic repetition of a single series using a simple methodology. It has been verified that such a method would result in desirable statistical properties [4]. Also, it has been verified that if the monotonic and low interactions assumptions are valid, the P&B design generates comparable results as the *full factorial design*. The detailed theoretical explanation behind this behavior is explained in [19].

The dimensions of the *P&B design matrix* depend on the number of configuration parameters, N . The base design matrix has X rows and $X - 1$ columns, where X is the next multiple of four greater than N , i.e., $X = \lfloor (N/4) + 1 \rfloor * 4$. For example, if $N = 7$, then $X = 8$, while if $N = 8$, then $X = 12$. If $N < (X - 1)$, then the 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 design matrix* are simply ignored. The recommendations of the “+1” and “-1” parameter value settings for $X = 8, 12, 16, \dots, 96, 100$ experiments are given in [19]. The first row of the *P&B design matrix* is selected based on those recommendations according to the value of X . The rest $(X - 1)$ rows of the *P&B design matrix* are constructed by right cyclic shifting of the immediate preceding row. All entries of the Exp_X -th row of the P&B design matrix are set to “-1”. The columns 2-8 in the first eight experiments (Exp_1 - Exp_8) of the Table 1 indicates the base *P&B design matrix* for $N=7$.

An improvement of the base *P&B design* methodology is the *P&B design with foldover* [18]. The *foldover* helps to quantify the two parameter interactions more accurately. However, it requires X additional experiments. The additional rows in the *P&B design matrix* are constructed by reversing the sign of the top X rows matrix entries. The last eight rows (Exp_9 - Exp_{16}) of Table 1 gives the additional design matrix entries for the *foldover* for $N=7$. Experiments are conducted by setting up the values of the configuration parameters according to the *P&B design matrix* and response time is recorded to estimate the effect of each parameter.

	P&B Effect							COV of Execution Times
	P_1	P_2	P_3	P_4	P_5	P_6	P_7	
Q1	109	79	167	193	21	25	177	0.55
Q2	61	161	9	143	39	185	109	0.32
Q3	0.40	0.80	0.00	0.40	0.40	0.00	0.40	0.01

Table 2: The P&B effects for the queries Q1, Q2, and Q3.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7
Q1	0.56	0.41	0.87	1.0	0.11	0.13	0.92
Q2	0.33	0.87	0.05	0.77	0.21	1.0	0.59

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

3 Effect Estimation of the Configuration Parameters

This section describes how to use the *P&B design* methodology described in Section 2 to estimate the effects of configuration parameters for each query of the given workload. The effect of each configuration parameter is calculated by multiplying the corresponding “+1” or “-1” of that parameter in the Exp_i -th row of the *P&B design matrix* with the query execution time of the i -th experiment, and summing up the products across all rows of the design matrix. The absolute value of the net effect is used in the subsequent analysis.

For illustration, suppose we estimate the P&B effects of a query workload consisting of three queries Q1, Q2, and Q3 as listed in Table 1. We have seven configuration parameters, P_1 to P_7 . In this example, we assume that *foldover* is used, therefore we conduct 16 experiments. The specification of the parameter values that need to be used in all 16 experiments are given in columns 2-8 and rows Exp_1 - Exp_{16} of Table 1. The net effect of the first parameter P_1 for query Q1 is calculated by multiplying the entries in the second column with the entries in the ninth column and summing up across all 16 rows (Exp_1 - Exp_{16}). For query Q1, the net effect of the parameter P_1 is estimated as: $Effect_{P_1} = abs((+1*34) + (-1*19) + \dots + (-1*51) + (+1*76)) = abs(-109) = 109$. Similarly, the net effect of the second parameter P_2 for query Q1 is calculated by multiplying the entries in the third column with the entries in the ninth column and summing across all 16 rows (Exp_1 - Exp_{16}), and so on. Table 2 gives the net P&B effects of all seven parameters for the queries Q1, Q2, and Q3.

The last column of Table 2 gives the *coefficient of variation (COV)* of the response time across all experiments for queries Q1, Q2, and Q3. COV is defined as the ratio of the *standard deviation* to the *average* execution time. A very low COV value indicates that all effects are essentially the same, i.e., the query performance will not be affected by the change in configuration parameters settings. In general, if the COV is less than 0.05, we can safely ignore the effects and mark the corresponding query as tuning insensitive. In the illustrative example, query Q3 is tuning insensitive as its COV is 0.01.

4 Exploiting the Configuration Parameters Effects

Once we have the P&B effects of the configuration parameters for all workload queries, we can use these estimated effects to: (1) rank the configuration parameters for the entire workload based on their relative impact on the DBMS performance, and (2) select a compressed query workload that preserves the fidelity of the original query workload. We describe these two methodologies in detail in the following subsections. Throughout this section, we will use the query workload in Table 1 as a running example.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7
Q1	3	4	1	1	7	7	1
Q2	5	1	7	2	6	1	3

Table 4: Ranking of the configuration parameters for the queries Q1 and Q2.

4.1 Ranking the Configuration Parameters for a Query Workload

Ranking configuration parameters for a query workload consists of two steps. First, we rank the parameters for each tuning sensitive query of the workload based on the relative magnitude of their P&B effects. Second, rankings of individual tuning sensitive queries are combined to estimate the overall ranking of a parameter for the workload. The queries which are insensitive to parameter tuning are not included in the workload ranking calculation. Therefore, we do not consider query Q3.

To rank the configuration parameters for a tuning sensitive query, the estimated P&B effects are normalized with respect to the maximum effect and the range of normalized effects are divided into N buckets, where N is number of configuration parameters. A parameter is assigned to the rank i if its normalized effect falls into the i -th bucket range. In the continuing example, we have seven parameters. Therefore the range for the first, second, third, fourth, fifth, sixth, and seventh buckets are, $[1, 0.86)$, $[0.86, 0.71)$, $[0.71, 0.57)$, $[0.57, 0.43)$, $[0.43, 0.29)$, $[0.29, 0.14)$, and $[0.14, 0.0]$, respectively. The normalized and rounded P&B effects for the queries Q1 and Q2 are listed in Table 3. For Q1, the rank of P_1 is 3 as its normalized P&B effect 0.56 falls into the third bucket. Similarly, the rank of P_2 is 4, and so on. The ranking of parameters for queries Q1 and Q2 are listed in Table 4. In the next step, ranks are summed across all tuning sensitive queries, averaged, and sorted in ascending order. The most important parameters will have the lowest cumulative rank. The average rankings of the parameters $P_1, P_2, P_3, P_4, P_5, P_6$ and P_7 of queries Q1 and Q2 as listed in Table 4 are 4.0, 2.5, 4.0, 1.5, 6.5, 4.0, and 2.0, respectively. Therefore, final ranking is 4, 3, 4, 1, 7, 4, and 2, respectively. Ranking indicates that P_4 is the most important configuration parameter, P_7 is the second most important configuration parameter, followed by $P_2, \{P_1, P_3, P_6\}$, and P_5 , in order. A detailed description of this methodology can be found in [10].

4.2 Compressing a Query Workload

To select a compressed query, all queries of the original workload are divided into two groups: tuning sensitive and insensitive. One query from the insensitive group is included in the compressed query workload, while the tuning sensitive group is further divided into subgroups based on the similarities of the effects of the configuration parameters. For each query, the effects are normalized to the maximum effect of the parameters for the corresponding query. Then, the Euclidean distance of the normalized effects among the queries is calculated to estimate a similarity score. If the Euclidean distance between the effects of the two queries is less than the user-chosen similarity threshold, we consider them as similar queries in terms of the performance bottlenecks and place them in the same group. Finally, one query is selected from each subgroup to include in the compressed query workload.

In the continuing example, the tuning sensitive group consists of queries Q1 and Q2; and the insensitive group consists of query Q3. The Euclidean distance between the normalized effects of queries Q1 and Q2 from Table 3 is $\sqrt{(0.56 - 0.33)^2 + (0.41 - 0.87)^2 + \dots + (0.13 - 1.0)^2 + (0.92 - 0.59)^2} = 1.36$. If the threshold of similarity score is 1.50, then we can consider queries Q1 and Q2 to be similar in terms of their impact of the performance bottleneck parameters. In the compressed query workload, we can include either query Q1 or query Q2. If we select query Q1, then the compressed query workload consists of queries Q1 and Q3. A detailed description of this methodology can be found in [21].

5 Experimental Results

All the experiments in this section are conducted in a machine with two Intel XEON 2.0 GHz w/HT CPUs, 2 GB RAM, and 74 GB 10,000 RPM disk. We use the TPC-H benchmark [2] and PostgreSQL [1] to demonstrate our methodologies. For demonstration, we use 22 read-only TPC-H queries, Q1-Q22, and data size of 1 GB. We consider only those PostgreSQL configuration parameters that are relevant to the read-only queries. The high and low values of each parameter are chosen in a range such that it will act as a monotonic parameter. A detailed information of the values used can be found in [21]. Furthermore, we have included some parameters, for example, `fsync` and `checkpoint_timeout`, which are not relevant for the read-only queries, yet they help in verifying that our method is working correctly. If their rankings or effects become low compared to other parameters, then it will give an indication that our method correctly identifies the performance bottlenecks.

The P&B effects of all configuration parameters are calculated using the *P&B design with foldover*. In order to identify the insensitive queries, COV of 0.05 is selected as a threshold. Out of 22 queries, the COVs of the queries Q4, Q6, Q7, Q10, Q11, Q12, Q14, Q15, Q17, Q18, and Q22 are found to be less than 0.05. These 11 queries formed one single group of tuning insensitive queries. The ranking of the parameters for tuning sensitive queries is listed in Table 5. For more detailed results of the estimated and normalized P&B effects, the readers are referred to [21]. Different parameters with the same rank indicates that they have similar effects on the performance. For query Q1, `work_mem` is the most important parameter, while other parameters do not have any impact on performance. For query Q2, `effective_cache_size` and `shared_buffers` are the first and sixth most important parameters, respectively, while other parameters do not have significant impact on performance. Similarly, the ranking of the parameters for other queries indicates the relative importance of corresponding parameter in the query performance.

The ranking of the parameters for the original TPC-H query workload consisting of tuning sensitive queries is listed in the first and second columns of Table 6. The results indicate that `work_mem` is the most important configuration parameter, `shared_buffers` and `effective_cache_size` are second most important parameters, followed by `cpu_operator_cost`, `random_page_cost`, and so on. The result also indicates that `fsync` and `checkpoint_time_out` do not appear in the top five most important parameters list. To verify that our results match with the decisions made by DBAs, we compare our parameter ranking against the PostgreSQL 8.0 Performance Checklist [5]. This checklist is a set of rules of thumb for setting up PostgreSQL server where it suggests the settings of configuration parameters that most DBAs will want to change. Among the parameters we are considering, according to this checklist there are six important parameters that need to be tuned, namely, `max_connections`, `shared_buffers`, `work_mem`, `maintenance_work_mem`, `effective_cache_size`, and `random_page_cost`. Four of these six parameters appear in our top five important parameters list. The differences between our result and this list are: (1) we find that the parameter `cpu_operator_cost` is an important one to our query workload and (2) the parameter `max_connections` appears to be less important to our workload as we do not consider concurrently running queries. Therefore, in general, our ranking methodology matches the general guidelines that are suggested for database tuning in addition to adding specific tuning decisions that match the given query workload.

To select a compressed query workload, we set a threshold of 0.5 for the Euclidean distance. At this threshold the 11 tuning sensitive queries form eight groups: {Q1, Q8, Q16}, {Q2, Q13}, {Q3}, {Q5}, {Q9}, {Q19, Q21}, and {Q20}. In the compressed workload, from each group we include the query which creates less perturbations in the original query workload ranking. In addition, we have to include one query from the insensitive group in the compressed workload. We select the query which has the largest execution time. Our compressed query workload includes queries Q2, Q3, Q5, Q8, Q9, Q18, Q20, and Q21. The new ranking for the compressed query workload is given in the third and fourth columns of Table 6. The result indicates that except `shared_buffers` and `effective_cache_size`, the ranking of the rest of the parameters is identical. `shared_buffers` is ranked second in the original query workload, while it is ranked third in the compressed query workload. On the other hand, `effective_cache_size` is ranked second in the original query workload, while it is ranked

Parameter	Q1	Q2	Q3	Q5	Q8	Q9	Q13	Q16	Q19	Q20	Q21
checkpoint_timeout	15	15	15	13	15	4	15	15	15	7	15
deadlock_timeout	15	15	13	15	13	4	15	15	15	7	14
fsync	15	15	15	15	15	2	15	15	15	6	15
max_connections	15	15	12	13	13	11	15	15	15	6	14
shared_buffers	15	6	15	15	15	12	7	14	1	6	1
stats_start_collector	15	15	15	15	15	11	15	15	15	7	14
cpu_index_tuple_cost	15	15	13	15	15	9	15	15	15	7	12
cpu_operator_cost	15	15	10	13	15	1	15	15	15	1	15
cpu_tuple_cost	15	15	13	13	14	8	15	15	15	7	14
effective_cache_size	15	1	11	13	15	1	1	15	15	6	13
geqo	15	15	9	13	12	10	15	15	15	7	14
maintenance_work_mem	15	15	11	15	11	10	15	15	15	6	12
random_page_cost	15	15	13	1	13	4	15	15	15	12	15
temp_buffers	15	15	11	15	15	3	15	15	15	7	15
work_mem	1	15	1	13	1	9	15	1	15	7	15

Table 5: Ranking of the configuration parameters for the tuning sensitive TPC-H queries.

Rank	Original Workload	Rank	Compressed Workload
1	work_mem	1	work_mem
2	effective_cache_size	1	effective_cache_size
2	shared_buffers	3	shared_buffers
4	cpu_operator_cost	3	cpu_operator_cost
5	random_page_cost	5	random_page_cost
6	geqo	6	geqo
6	maintenance_work_mem	6	maintenance_work_mem
6	deadlock_timeout	6	deadlock_timeout
6	temp_buffers	6	temp_buffers
10	max_connections	10	max_connections
10	cpu_tuple_cost	10	cpu_tuple_cost
10	fsync	10	fsync
10	checkpoint_timeout	10	checkpoint_timeout
14	cpu_index_tuple_cost	14	cpu_index_tuple_cost
15	stats_start_collector	15	stats_start_collector

Table 6: Ranking of the configuration parameters estimated by the original and compressed query workloads.

first in the compressed query workload. However, as long as the list of topmost important parameters does not change drastically, in reality it does not cause much impact in tuning activities.

6 Related Work

Major database vendors offer tools for tuning database physical design [9, 3, 22]. IBM DB2 provides a utility named `autoconfigure` for automatically selecting the initial values for the configuration parameters based on generic workload behavior [15]. Oracle Automatic Database Diagnostic Monitor (ADDM) tool possesses a holistic view of the database, identifies root causes of the performance bottlenecks, and estimates the benefits of eliminating performance bottlenecks [12]. In Microsoft SQL Server, most of the parameters can be configured either through Enterprise Manager or with the T-SQL `sp_configure` command [11]. However, none of the current tools rank the configuration parameters based on their impact on the DBMS performance.

Two major techniques for query workload compression are proposed in the literature. The first technique groups SQL statements based on the accessed tables and join columns [7]. The second technique focuses on the most complex and costly queries in the workload and ignore other queries [22]. In contrast, our proposed workload compression methodology selects subset workload based on similarities of the performance bottlenecks of the configuration parameters.

7 Conclusion

In this article, we have proposed methodologies for ranking configuration parameters and selecting a compressed query workload based on the impact of configuration parameters on the DBMS performance for a given query workload. Our proposed methodologies are quite generic and can be applied to non-database systems as well. These methodologies will greatly help DBAs of all knowledge levels to prioritize tuning activities and to reduce their valuable time and efforts. In the future, we are planning to perform the following extensions: 1) validating the assumptions behind calculating parameter effects, and 2) suggesting the appropriate values of the configuration parameters using the estimated P&B effects.

8 Acknowledgements

This work was supported in part by National Science Foundation grant no. CCF-0621462 and CCF-0541162, the University of Minnesota Digital Technology Center Intelligent Storage Consortium, and the Minnesota Supercomputing Institute.

References

- [1] PostgreSQL DBMS Documentation. <http://www.postgresql.org/>.
- [2] Transaction Processing Council. <http://www.tpc.org/>.
- [3] S. Agrawal, S. Chaudhuri, L. Kollar, A. Marathe, V. Narasayya and M. Syamala. Database Tuning Advisor for Microsoft SQL Server 2005. In *y of VLDB*, 2004.
- [4] T. Allen. *Introduction to Engineering Statistics and Six Sigma: Statistical Quality Control and Design of Experiments and Systems*. Springer, 2006.
- [5] J. Berkus. Power PostgreSQL: PostgreSQL Performance Pontificated. <http://www.powerpostgresql.com/PerfList/>.
- [6] D. Cappucio, B. Keyworth and W. Kirwin. The Total Cost of Ownership: The Impact of System Management Tools. Strategic Analysis Technical Report, Gartner Group, Stamford, CT, 1996.
- [7] S. Chaudhuri, A. K. Gupta and V. Narasayya. Compressing sql workloads. In *Proc. of SIGMOD*, 2002.
- [8] S. Chaudhuri and G. Weikum. Rethinking Database Architecture: Towards s Self-tuning RISC-style Database System. In *Proc. of VLDB*, 2000.
- [9] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait and M. Ziauddin. Automatic SQL Tuning in Oracle 10g. In *Proc. of VLDB*, 2004.
- [10] B. Debnath, D. Lilja and M. Mokbel. SARD: A Statistical Approach for Ranking Database Tuning Parameters. In *Proc. of 3rd Intl. Workshop on Self-Managing Database Systems*, 2008.
- [11] S. DeLuca, M. Garcia, J. Reding and E. Whalen. *Microsoft SQL Server 7.0 Performance Tuning Technical Reference*. Microsoft Press, March 2000.
- [12] K. Dias, M. Ramacher, U. Shaft, V. Venkataramamani and G. Wood. Automatic Performance Diagnosis and Tuning in Oracle. In *Proc. of CIDR*, 2005.
- [13] C. Garry. Who's Afraid of Self-Managing Databases? <http://www.eweek.com/article2/0,1895,1833662,00.asp>, June 30, 2005.
- [14] Hurwitz Group. Achieving Faster Time-to-Benefit and Reduced TCO with Oracle Certified Configurations. March, 2002.
- [15] E. Kwan, S. Lightstone, A. Storm and L. Wu. Automatic Configuration for IBM DB2 Universal Database. In *IBM Performance Technical Report*, January 2002.
- [16] 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), 2006.
- [17] D. Lilja. *Measuring Computer Performance A Practitioner's Guide*. Cambridge University Press, 2000.
- [18] D. Montgomery. *Design and Analysis of Experiments*. Wiley, 2001.
- [19] R. Plackett and J. Burman. The Design of Optimum Multifactorial Experiments. In *Biometrika Vol. 33 No. 4*, 1946.
- [20] A. Rosenberg. Improving Query Performance in Data Warehouses. <http://www.tdwi.org/Publications/BIJournal/display.aspx?ID=7891>, 2005.
- [21] J. Skarie, B. Debnath, D. Lilja and M. Mokbel. SCRAP: A Statistical Approach for Creating Compact Representational Query Workload based on Performance Bottlenecks. In *Proc. of IISWC*, 2007.
- [22] 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 *Proc. of VLDB*, 2004.