An Adaptive Hybrid Controller for DBMS Performance Tuning

Sherif Mosaad Abdel Fattah, Maha Attia Mahmoud, Laila Abd-Ellatif Abd-Elmegid
Department of Information Systems
Faculty of Computers and Information
Helwan University
Helwan, Egypt

***Abstract*—Performance tuning process of database management system (DBMS) is an expensive, complex and time consuming process to be handled by human experts. A proposed adaptive controller is developed that utilizes a hybrid model from fuzzy logic and regression analysis to tune the memory-resident data structures of DBMS. The fuzzy logic module uses flexible rule matrix with adaption techniques to deal with fluctuations and abrupt changes in the operation environment. The regression module predicts fluctuations in operation environment so the controller can take former action. Experimental results on standard benchmarks showed significant performance enhancement as compared to built-in self-tuning features.**

Keywords—automatic database tuning; fuzzy logic; adaptive controller; regression; self-tuning; DBMS

I. INTRODUCTION

Database management system performance tuning is a complex process with multiple objectives and tuning parameters. To know how to enhance such a process we need first to understand its characteristics and components. DBMS performance tuning can be generally described as a group of six activities to optimize the performance of a database[1].

Design Tuning tries to follow DB design best practices and normalizing DB tables to reveal un-optimized design issues that can degrade the performance. *SQL Tuning* tries to enhance the formulation of SQL statements to optimize the execution of the queries. *Memory Tuning* deals with allocating suitable values to the DB memory-resident data structures such as Shared Pool, Buffer Cache or Redo Log Buffer. *I/O Tuning* deals with I/O read/write anomalies such as disk fragmentation levels and tries to adjust its parameters for performance enhancements. *Connection Tuning* monitors network bandwidth and traffic and tries to optimize communication. *OS Tuning* investigates the system parameters and tries to adjust operation parameters such as virtual memory amount or size of memory page to enhance the performance of the DB environment.

DBMS performance tuning isn't an atomic process and it has a dynamic nature which makes its management harder and expensive due to need for an expert Database Administrator (DBA). The changes in the operation environment such as number of concurrent users, queries load, available memory space or network bandwidth can tend any performance tuning model to be unfeasible and outdated quickly if it can't adapt with these changes.

The term self-tuning databases[2] was coined for the aim of having a database that can learn and adapt with its environment with low or no interference from the human experts. To achieve this goal we have to depend on dynamic and adaptive control techniques such as fuzzy logic and nonlinear regression analysis.

In this paper, an adaptive hybrid controller (AHC) for DBMS memory-resident data structures is introduced. The controller utilizes hybrid model derived from fuzzy logic and regression analysis. The controller periodically monitors and feeds performance indicators of DBMS memory-resident data structures into fuzzy logic engine. The fuzzy logic engine fires corrective actions rules. The regression analysis module provides the controller with the ability to predict abrupt changes in the operation environment to further enhance the tuning process.

The rest of this paper is organized as follows: Section II describes preliminary concepts. Section III reviews previous work. Section IV introduces our proposed solution. Section V illustrates the experimental evaluations and results. Finally, Section VI concludes the paper and lists future work.

II. BACKGROUND

A. DBMS Memory-Resident Data Structures

DBMS memory resident data structures play a critical role in the process of tuning the DBMS performance. As it may decrease/increase the time and memory needed to execute queries and transaction on the database. There are three common data structures in any modern DBMS; Redo Log Buffer, Shared SQL Pool and Data Block Buffer [3] we are going to introduce the Data Block Buffer as it is the focus of this research in the following section.

The data block buffer cache (DBB) is the space reserved in memory for holding data blocks. The larger the DBB parameter value, the more memory is available for holding data blocks. The actual size of the DBB in bytes is computed as follows:

$$DBB = DB\_BLOCK\_BUFFERS \times DB\_BLOCK\_SIZE \quad (1)$$

The efficiency of the cache is measured by a metric called the data block buffer hit ratio (DBB-HR) that records the percentage of times a requested data block is available in the cache out of the total number of requests. When a data block is read in cache, it is called a logical read (LR). When the block is read from disk, it is called a physical read (PR).

$$(DBB\ Hit\ Ratio) = \frac{LR-\mathrm{PR}}{LR} \qquad (2)$$

For less than 20 concurrent users DBB-HR should be between 91% and 94%. Otherwise, it should exceed 94% in a healthy DBMS instance [3].

B. Fuzzy logic

Fuzzy logic (FL) mimics the ability of human brain in the usage of reasoning modes that are approximate rather than exact [4]. In traditional computing models, decisions are based on certainty and vigor but, this carries a cost of failure to deal with non-linear and complex problems that involve uncertainty in its characteristics. Examples to those problems can be, understanding human speech, sloppy handwriting, summarizing text or recognizing images.

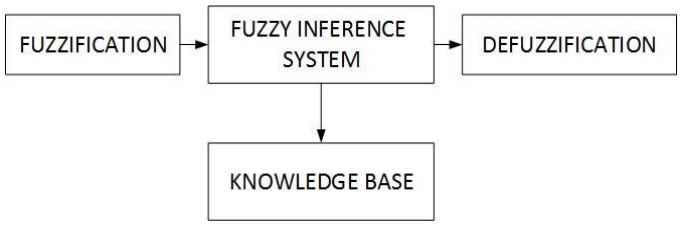

Fig. 1. Fuzzy Control Process

With Fuzzy Logic, decision rules are mapped to words rather than numbers. Computing based on words rather than exact number has tolerance to deal with uncertainty[5]. Broadly described, FL working scenario involves converting inputs of the problem from numerical nature (exact) to word based (approximate) nature in a process called Fuzzification. Then, the fuzzified inputs are supplied to the fuzzy inference engine which contains the inference rules to reach conclusions. Finally, the outputs are transformed from their approximate nature to an exact nature in a process called Defuzzification.

C. Regression Analysis

Regression analysis is a statistical technique used to predict the value of dependent variable (Y) given the values of independent variables (X1 … Xn)[6]. If the relation between the dependent and the independent variables is following a linear equation it is called linear regression and it can be represented by the following equation[7]:

$$Y = a + b_1 X_1 + \cdots + b_n X_n \qquad (3)$$

Where, *a* is intercept (the value of Y when x =0) and set (b_1 … b_n) represents slope of the line according to multiple (n) dimensions [X_1… X_n]

If the relation between the dependent and the independent variables is following a non-linear equation it is called non-linear regression. There are multiple models for nonlinear regression for example exponential model, power model or polynomial model[8].

III. Releated Wrok

The work in databases performance tuning started from decades and has been refined many times starting from the relational databases design concepts such as normalization forms and relational constraints to self-tuning databases ideas.

Ways in databases design tuning such as index pruning table and materialized views were addressed in [9][10]. Physical database tuning and the use of self-healing performance tuning methodologies were introduced in [11][12]. In [13], an modular approach was presented for providing self-healing database functionalities.

Each module in the system is assigned to a specific monitoring handler. In [14], a new way for physical data file organization based on search queries was proposed. Search queries are used to cluster similar records and to store them in one cluster block. So, I/O operations can be optimized in the physical layer. [1] Introduced a statistical approach to rank and evaluate the effect of database performance tuning parameters. In [15], operation research (OR) techniques were used to probe the SQL queries to optimize database logical design structures (schema) such as indexes or materialized views. [16] Introduced a neural networks based controller. Data mining techniques were used to analyze the database's log file to extract operation features.

Then, the result is used to train the neural network for controlling database's buffer cache levels. In [17], a fuzzy logic controller was introduced to tune the performance of web servers in terms of request response-time over multiple service level classes. Each Service level class will be assigned response-time level. The controller task is to maintain those service levels of response-time for each class when the server is heavy loaded. The work in [17] was extended in [18] to manage configuration of virtual machines on cloud-computing environments according to user's quality of service classes.

IV. Contribution

This research proposes a controller that employs hybrid criteria between fuzzy logic and regression analysis to adaptively tune the size of DBMS memory-resident data structures. The following figure describes the main components of the proposed controller:

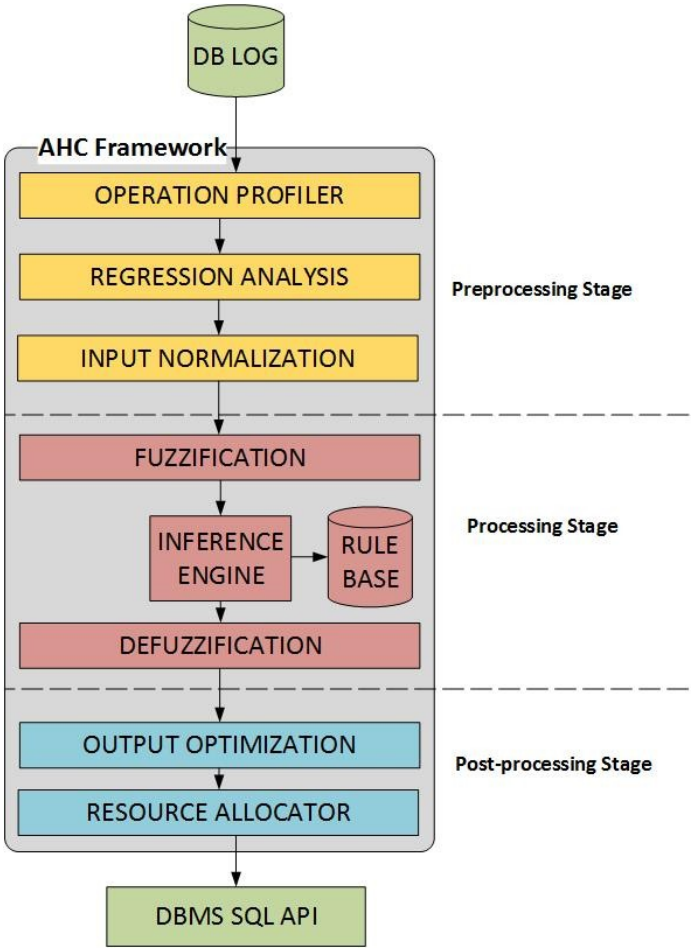

Fig. 2. Proposed System Architecture

In the next section we are going to explore AHC framework in a modular approach.

A. Preprocessing Stage

1) Operation Profiler

That module is responsible for collecting the current error values for each configured performance parameter. Performance parameters are configured in XML file. The DB admin configures the set of performance parameters along with their reference values. The error calculation depends on the current parameter value and its reference value equated as follows:

$$e(res) = \begin{cases} \frac{c(res)-r(res)}{r(res)}, & 0 \le c(res) \le 2r(res); \\ 1 \quad , & c(res) > 2r(res); \end{cases} \quad (4)$$

c(res) stands for the current resource value, *r(res)* for its reference value and *e(res)* for the error value.

2) Regression Analysis

Regression module is activated after a configured number of tuning cycles to collect sufficient amount of data. Regression type - linear or nonlinear - can be configured by the DB admin. For linear regression equation (5) is utilized to calculate the next value for the performance parameter. For, nonlinear regression the polynomial regression[8] is utilized using the following equation:

$$y = a + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k \quad (5)$$

As, a is the intercept and β is the regression coefficient for variables $X_1 \ldots X_k$ and y is the value to predict. Regression module is used to predict next error value. The input error value is the average between the current and predicted error values to help the controller to deal with abrupt changes in the environment.

3) Input Normalization

The input error value is normalized to avoid overshooting resource allocation due to peaks or dynamic resources changes. The error *e(res)* and error difference $\Delta_{e(res)}$ are normalized using the following equations [17]:

$$N_e(i+1) = |(1-\gamma) * N_e(i) + \gamma * e(res)| \quad (6)$$

$$N_{\Delta e}(i+1) = |(1-\gamma) * N_{\Delta e}(i) - \gamma * \Delta e(res)| \quad (7)$$

N_e stands for the normalized error, $N_{\Delta e}$ for normalized error difference and γ for the constant weight which equals to 0.8. This normalization technique homogenizes the current error value with its past values while, giving more weight for the current one. Note that the actual input to the resource controller module is $N_e * e(res)$ and $N_{\Delta e} * \Delta e(res)$. The sign is positive in equation (6) as error values take different signs in fluctuations and it is negative in equation (7) as the values of error difference take the same sign in fluctuations.

B. Processing Stage

1) Fuzzification

The input values *e(res)* and $\Delta_{e(res)}$ are fuzzified using triangular membership functions[5]. In triangular membership function the membership is calculated according to equation (8).

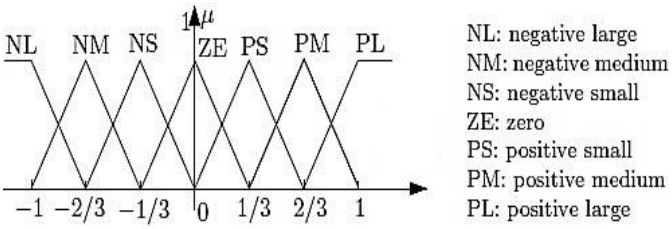

Fig. 3. Input Membership functions Graph

$$F(x) = \max\left(\min\left(\frac{x-a}{b-a}, \frac{c-x}{c-b}\right), 0\right) \quad (8)$$

Where a, b (center) and c are triangle membership vertices from left to right.

Table I. INPUT MEMBERSHIP FUNCTIONS

Membership Function	a	b	c
Negative Large (NL)	-∞	-1	-2/3
Negative Medium (NM)	-1	-2/3	-1/3
Negative Small (NS)	-2/3	-1/3	0
Zero (ZE)	-1/3	0	1/3
Positive Small (PS)	0	1/3	2/3
Positive Medium (PM)	1/3	2/3	1
Positive Large (PL)	2/3	1	∞

2) Inference Engine

The utilized inference mechanism is derived from [17] [18] and adjustment value is calculated using equation (9):

$$U_{(i+1)} = U_{(i)} + \alpha * Adj_{max} * \Delta U_{(i)} \quad (9)$$

As $U_{(i+1)}$ represents the adjustment for the current resource value, $U_{(i)}$ the current value of the resource in interval (i),

α , Adj_{max} are the output optimization factors and $\Delta U_{(i)}$ the inference engine output. α , Adj_{max} are illustrated in output optimization section.

The following algorithm specifies steps for calculating $\Delta U_{(i)}$

Input: e(res) , Δe(res).
Output: $\Delta U_{(i)}$
Steps:
for each mf ϵ Input_MembershipFunctions
x = TriangularMembership(e(res), mf)
y = TriangularMembership(Δe(res), mf)
if x > 0 THEN
add x to e(res)_MembershipFunctions
if y > 0 THEN
add y to Δe(res)_ MembershipFunctions
rules = combine(e(res)_MembershipFunctions, Δe(res)_ MembershipFunctions)
for each r ϵ rules
match = getMatrixMatch(r)
center = Center(match)
μ_r = MIN(r)
add center to Centers
add μ_r to WeightSet
Output = CenterOfGravity(Centers, WeightSet)

The following table represents the proposed rule matrix. Columns and rows represent membership functions of Δe(res) and e(res) respectively. Each element in the table construct a rule for example, (NL,NL) > PL

Table II. RULE MATRIX FOR $\Delta U_{(i)}$

ΔU(i)		Δe(res)						
		NL	**NM**	**NS**	**ZE**	**PS**	**PM**	**PL**
e(res)	**NL**	PL	PL	PL	PL	PM	PS	ZE
	NM	PL	PL	PM	PM	PS	ZE	NS
	NS	PL	PL	PM	PS	ZE	NS	NM
	ZE	PM	PM	PS	ZE	NS	NM	NM
	PS	PM	PS	ZE	NS	NM	NL	NL
	PM	PS	ZE	NS	NM	NL	NL	NL
	PL	ZE	NS	NM	NL	NL	NL	NL

Suppose for e(res) = 0.075 and Δe(res) = 0.3 the following membership functions have non zero membership

ZE (e(res)) = 0.6 , PS(e(res)) = 0.3

And for Δe(res) PS(Δe(res)) = 0.75

Then, combinations are generated from previous rules:

Rule1 (ZE,PS) with μ = MIN(0.6,0.75) = 0.6

Rule2 (PS,PS) with μ = MIN(0.3,0.75) = 0.3

Rule1 gives us the match NS from the rule matrix which has a center = $-\frac{1}{3}$

Rule2 gives us the match NM from the rule matrix which has a center = $-\frac{2}{3}$

3) Defuzzification

In defuzzification, the output $\Delta u(i)$ is calculated using the Center of Gravity equation (10) [5]:

$$\Delta U_{(i)} = \frac{\sum_{n}^{i=1} center(i) * \mu(i)}{\sum \mu(i)} \quad (10)$$

According to our example $\Delta U_{(i)}$ will equal $\frac{-4}{9}$

C. Post-processing Stage

1) Output Optimization

The output optimization factor (α) is used to handle process delay during resource allocation. It is summarized as the time between sending the new adjustment of a resource and the time the resource value is actually updated[19].

α is calculated with the same criteria as $\Delta U_{(i)}$ but with different membership functions and rule matrix.

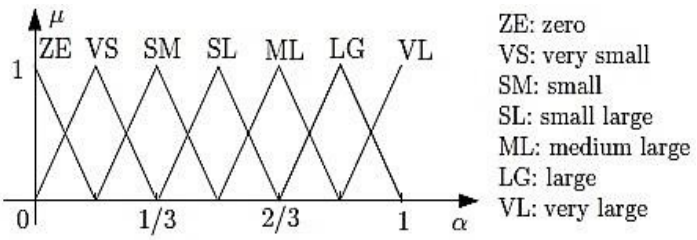

Fig. 4. α membership functions Graph

Table III. α MEMBERSHIP FUNCTIONS

Membership Function	a	b	c
Zero (ZE)	-∞	0	1/6
Very Small (VS)	0	1/6	1/3
Small Medium (SM)	1/6	1/3	1/2
Small Large (SL)	1/3	1/2	2/3
Medium Large (ML)	1/2	2/3	5/6
Large (LG)	2/3	5/6	1
Very Large (VL)	5/6	1	∞

The value of α is used to speed up or slow down the change magnitude of the resource value. In fluctuations periods it is set to small value to prevent overshooting in adjustment. When the current value is going away from the reference value it is set to relative large value to invert the change.

Table IV. RULE MATRIX FOR A

ΔU(i)		Δe(res)						
		NL	NM	NS	ZE	PS	PM	PL
e(res)	NL	VL	VL	VL	SM	VS	VS	ZE
	NM	VL	VL	LG	SL	SM	SM	SM
	NS	VL	VL	LG	ML	VS	SM	SL
	ZE	LG	ML	SL	ZE	SL	ML	LG
	PS	SL	SM	VS	ML	LG	LG	VL
	PM	SM	SM	SM	SL	LG	VL	VL
	PL	ZE	VS	VS	SM	VL	VL	VL

Adj_{max} is the max adjustment value that can be allocated. It is calculated as follows:

$$Adj_{max} = \left|\frac{c}{2} * e(res)\right| \qquad (11)$$

Where c is the current resource value. This equation is based on heuristic control rule[19] which states that the max resource adjustment shouldn't exceed half of the current resource value for stability of the system and to be proportional to the current error value for adaptability of the system.

2) Resource Allocator

This module concludes the work of the control cycle by sending the adjustment value(s) to the DBMS API for allocation using SQL commands.

V. EVALUATION AND EXPERIEMENTS

TPC-C and TPC-H benchmarks[20] are used to conduct the evaluation on data block buffer data structure; TPC-C is an online transaction processing (OLTP) benchmark. It involves a mix of five concurrent transactions of different types and complexity. TPC-H is a decision support benchmark. It consists of ad-hoc and concurrent queries. The operation environment for the experiments runs on Windows server 2008 with ORACLE 10g database server installed. Our proposed system is deployed as windows service. The user load is defined as 20 concurrent users that start with 2 concurrent users in the first transaction cycle and increase gradually by 2 until reaching 20 in the following run cycles. The tuning cycle period for the controller is set to 30 seconds (defined by trial and error to pose the minimum overhead on the DBMS performance while keeping track of workload changes). Regression is activated after 30 tuning cycles. It is configured to be nonlinear with variables; number of users per cycle and number of transactions per minute.

Figure 5 shows results for conducting TPC-C benchmark. The average response for the DBMS without tuning was 87 ms while the average response time after AHC tuning was 46 ms with 52% better than without tuning.

Figure 6 shows results for conducting TPC-H benchmark. The average response for the DBMS without tuning was 93.1 ms while the average response time after AHC tuning was 46.5 ms with 49% better than without tuning.

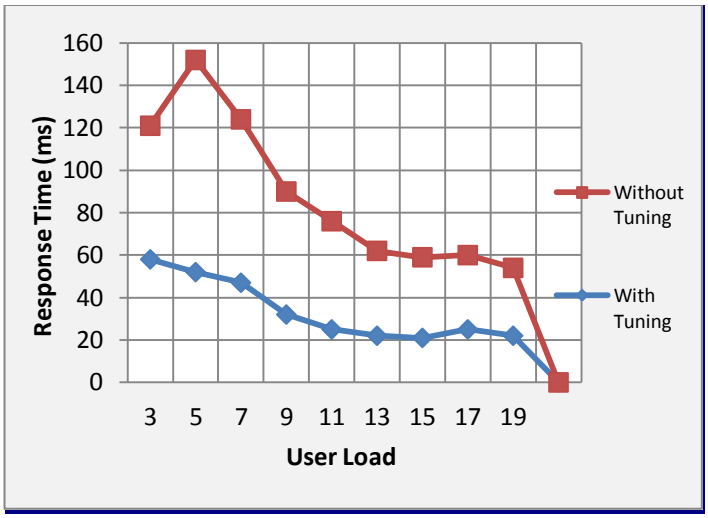

Fig. 5. TPC-C benchmark, comparing response time with and without tuning

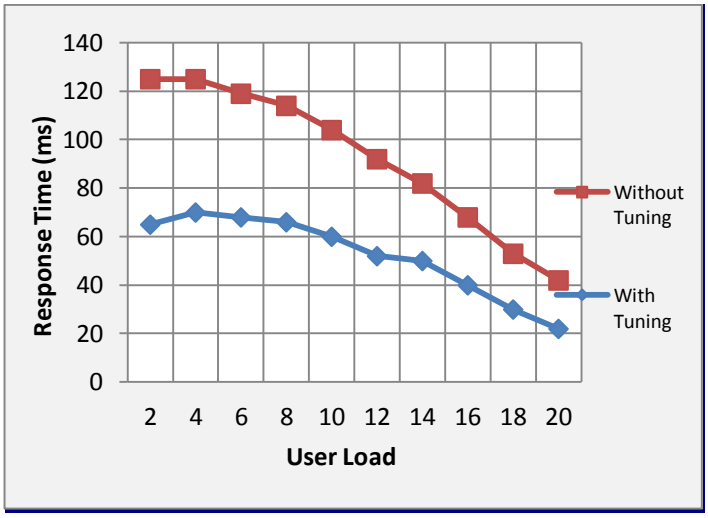

Fig. 6. TPC-H benchmark, comparing response time with and without tuning

VI. CONCLUSION AND FUTURE WORK

This paper proposes an adaptive hybrid controller (AHC) for tuning DBMS performance based on its resident data structures. AHC is featured by both generalization and adaptability. AHC generalization is achieved in three ways. First, operation profiler and resource allocator are developed to deal with any type of DBMS using Microsoft generic ADO API[21]. Second, the proposed rule matrices can be totally configured and adjusted using XML configuration file to adapt with different workload scenarios in a generic way. Third, regression analysis module allows the system to take former action with fluctuations and abrupt changes in the operation environment and workload. It can be configured by the DBA according to each resource case giving more flexibility to deal with different scenarios. AHC is adaptable in two ways. First, input normalization module normalizes inputs with its past values to reveal the fluctuations effect. Second, output optimization factors deal with process delay effect and scale the output to prevent down or over shooting in resources allocation.

Future work can include covering other memory-resident data structures of DBMS. Using machine learning techniques such as Neural Networks to equip our controller with the ability to learn the characteristics of its operation environment and to dynamically adjust its membership functions and rule matrices according to characteristics of work load on the operation environment.

References

[1] B. K. Debnath, D. J. Lilja And M. F. Mokbel, "Sard: A Statistical Approach For Ranking Database Tuning Parameters," Ieee 24th International Conference, 2008.

[2] R. V. Nehme, "Dtabase, Heal Thyself," Data Engg.Workshop, April 2008.

[3] S. S. Mittra, Database Performance Tuning And Optimization Using Oracle, Springer, 2003.

[4] J. H. Lilly, Fuzzy Control And Identification, Wiley, 2010.

[5] J. Jantzen, Foundations Of Fuzzy Control, John Wiley And Sons, 2007.

[6] A. O. Sykes, "An Introduction To Regression Analysis," [Online]. Available: Http://Www.Law.Uchicago.Edu/Files/Files/20.Sykes_.Regression.Pdf.

[7] A. Cottrel, "Regression Analysis: Basic Concepts," 2011. [Online]. Available: Http://Users.Wfu.Edu/Cottrell/Ecn215/Regress.Pdf.

[8] C. Ritz And J. C. Streibig, Nonlinear Regression With R, Springer, 2008.

[9] S. Agrawal, S. Chaudhuri And V. Narasayya, "Automated Selection Of Materialized Views And Indexes For Sql Databases," Microsoft Research, 2007.

[10] Surajit, Chaudhuri; Vivek, Narasayya;, "Self Tuning Database Systems : A Decade Progress," Microsoft Research, 2007.

[11] K. Philip, "Elements Of The Self-Healing System Problem Space," Ieee Data Engineering Bulletin, 2004.

[12] P. Liu, "Design And Implementation Of Self Healing Database System," Ieee Conference, 2005.

[13] R. V. Nehme, "Database, Heal Thyself," Data Engg. Workshop, 2008.

[14] B. G. L. A. A. Kitsopanidis, "Enhancing Database Retrieval Performance Using Record Clustering," Citeseerx, 2007.

[15] A. N. Chen, "Robust Optimization For Performance Tuning Of Modern Database Systems," European Journal Of Operational Research 171, 2006.

[16] U. P. K. S. F. Rodd, "Adaptive Tuning Algorithm For Performance Tuning Of Database Management System," International Journal Of Computer Science And Information Security, Vol. 8, 2010.

[17] J. Wei And C.-Z. Xu, "Eqos: Provisioning Of Client-Perceived End-To-End Qos Guarantees In Web Servers," Ieee Transaction On Computer, 2006.

[18] J. Rao, Y. Wei, J. Gong And C.-Z. Xu, "Dynaqos: Model-Free Self-Tuning Fuzzy Control Of Virtualized Resources For Qos Provisioning," Ieee Nineteenth Ieee International Workshop On Quality Of Service, 2011.

[19] F. G. Shinskey, Process Control Systems: Application, Design, And Tuning., Mcgraw-Hill, 1996.

[20] "Tpc," [Online]. Available: Http://Www.Tpc.Org/Tpcc/.

[21] Msdn, "Ado.Net," Microsoft, [Online]. Available: Http://Msdn.Microsoft.Com/En-Us/Library/E80y5yhx(V=Vs.110).Aspx.

[22] A. M. Brown, "A Step-By-Step Guide To Non-Linear Regression Analysis Of Experimental Data Using A Microsoft Excel Spreadsheet," Computer Methods And Programs In Biomedicine , Vol. 65, P. 191–200, (2001).