Chapter 6

Adaptive Self-tuning : A Neuro-Fuzzy approach

6.1 Introduction

The adaptive techniques presented in the previous two chapters have shown how the performance-tuning of DBMS can be done effectively. Each of these methods have their own advantages and disadvantages with their respective implementation difficulty, performance advantage etc. The idea behind proposing a hybrid tuning architecture is to leverage the advantages of both and evaluate its performance. Since, DBMS software is a multi-module software comprising of several layers of processing components [16] for processing the end user queries, a multi-model self-tuning system is desirable. The end-user queries are parsed for their syntax and semantics, and then given to an optimizer for generating the most efficient execution plan. The Database engine then generates appropriate read/write commands with the execution plan as input.
The data blocks are read from the disk using the APIs(Application Program Interfaces) of the Operating System and the data is then presented to the end-user. In such a complex query execution environment, more than one method of estimating the extent of tuning to be carried is most appropriate. Hence, a Neuro-Fuzzy based tuning architecture is presented that has a much better performance as compared to the self-tuning feature of the DBMS.

6.2 Neuro-Fuzzy Tuning Architecture

The autonomic computing framework of IBM provides well defined basis for building self-optimizing systems using novel techniques by adding appropriate self-tuning techniques in the MAPE feed control loop. As can be seen from figure 6.1, the sensor inputs and effector outputs are the same as the self-tuning techniques presented in the previous two chapters. The Plan and Execute modules now have two techniques to estimate the extent of tuning required. The neural network uses its learning ability based on the training dataset to compute the tuning parameter values. The FIS uses the fuzzy rules to act on the fuzzified inputs to estimate the fuzzy values of the tuning parameters.

These values are defuzzified to generate the actual sensor values to be used for tuning. As we have seen before, the performance of Neural network based tuning is better as compared to Fuzzy based approach, it is given a 10% lower weightage in computing the final tuning parameter values. Hence, the output of the aggregator is weighted average of the inputs in the ratio of 60:40. The sensor inputs namely, BHR(Buffer-Hit-Ratio), N(User-load) and DBS(Database size) are the same for both the tuning modules. However, they derive their knowledge-base from their respective knowledge components. The Neural network module derives its knowledge from the training dataset and the fuzzy module from its fuzzy rule base.

Following section describes the details of the Neuro-Fuzzy control system and also the working of the tuning algorithm.

6.2.1 Neuro-Fuzzy tuning architecture

Figure 6.2 shows the details of the Neuro-fuzzy tuning architecture. The DBMS performance degradation indicators namely User-load(N), Buffer-Hit-Ratio(BHR) and Database Size(DBS) are extracted with the help of Parameter extraction module. These values are then given to pre-processing module that maps the actual sensor values to a set of values in the range [-1 to +1].
This step is necessary for some of the Neural network models such as Feedforward Backpropagation to generate stable output values. A reverse process is to be carried in the output stage for getting the real values from the scaled down output values.

The Neural network module comprises of a 4-layer network with the 3 nodes in the input, 3 nodes in the output and [1] [70] nodes in the hidden layers. The number of layers and the nodes in the hidden layers are slightly increased to improve the estimated values of the
tuning parameter as the weightage of this module is 40% in the output. To take advantages of both the methods presented in section 4.1 and 4.2, a hybrid approach namely, the Neuro-Fuzzy controlled tuning architecture is proposed. This comprises of a module that computes the input parameters for the Neuro-Fuzzy modules, namely the Buffer-Hit-Ratio (BHR), number of Active users, and the Database Size. The Fuzzy module consults a rule base and acts upon the fuzzified inputs to generate the estimated values of the tuning parameters. A properly trained Neural network estimates the values of the tuning parameters taking into account present user load, Buffer-Hit Ratio and Average Database size. The estimated values are then combined in the 60:40 ratio (Fuzzy-60:Neuro-40) to generate the final tuning parameter values.

6.2.2 Tuning Architecture

Calibrating the system for desired response-time is called performance tuning. The block schematic shows the functional blocks that make up the entire control system, external to the DBMS. As can be seen from the architecture, the three important input parameters namely, number of users (nUsers), Buffer Hit Ratio (BHR) and Database size (DBS) are fed to both the Neural and Fuzzy systems to estimate the appropriate size of the tuning parameters. The neural network is trained with an appropriate training data set derived from experimental observations. Though there are several tunable parameters, only three parameters are chosen for tuning as they have significant impact on the response time and most importantly are dynamically alterable without the need to bounce back the DBMS.

6.2.3 Neural Network subsystem

Neural networks are best suited to handle complex systems that are intrinsically non-linear in nature. Neural networks fall under a broader subject called machine learning that deal with computer solutions which learn the complex relationship between the input and the output from empirical training data-set. Application machine learning technique is quite widespread [62] [55] [61] that include medical image analysis to predict the deceases precisely, finding near optimization solutions to NP-Hard problems especially in networking domain etc. Machine learning provides an effective method of predicting the estimated values of the tuning parameters from a given training dataset.

As shown in Fig. 6.3, Neural Network will have P inputs, a specified number of nodes in the hidden layer and one or more output nodes. The neural network used in this control architecture is a feedforward backpropagation network. The activation function used is
sigmoidal function for all the inner nodes. It is this function that gives the neural network the ability to learn and produce an output for which it is not trained. However, the neural networks need a well defined training data-set for their proper functioning. The output nodes have a pure-linear activation function to generate the final estimated values for a given test input data.

The Neural Networks work in phases. In the first phase, the network is trained using a well defined training set for a desired output. In the second phase a new input called test input is presented to the network that may or may not be part of the training data set and network produces a desired output that is most appropriate for the given input data. In this proposed setup, the first hidden layer has 12 nodes and second hidden layer has 8 nodes. The learning rate is set to 0.0001 for more accurate results.

6.3 Neuro-Fuzzy based Self-Tuning algorithm

The following algorithm describes the steps followed in estimating the new values of the tuning parameters, their aggregation and moderation.
ALGORITHM PerformanceTuneDB(BHR, DBS, N, wLoadType[])

// Computes the values of the tuning parameters Tp[i..n] based on Neuro-Fuzzy approach and
Moderates the computed values based on the Impact Factor of the respective tuning
parameter.//Input : Buffer Hit Ratio(BHR), Database Size (DBS), Number of active users (N)
and Workload Type (wLoadType) and TPN \rightarrow the number of tuning parameters, NWtype \rightarrow
Number of workload types.

// Output : New Tuning parameter values Tp[1..n]

Begin for i \leftarrow 1 to TPN do

    TpNN[i]=NN(BHR, DBS, N) \quad //Estimate using NN approach

    TpFL[i]=FIS(BHR,DBS, N) \quad //Estimate using Fuzzy Logic approach

next i

// Compute Weighted average of TpNN and TpFL

for i \leftarrow 1 to TPN do

    Tp[i]= 0.6*\text{TpFL}[i] + 0.4*\text{TpNN}[i] \quad // Estimated Tp by FL was better compared to NN

next i

performTuning(Tp[i]) \quad //Perform tuning action

end

The algorithm assumes that a Neural network is already setup and is trained by a valid
training dataset and also the fuzzy inference system is ready for use with appropriate fuzzy
rules in place. For each tuning parameter, the algorithm computes the new value using both the
methods and the final value of tuning parameter would be the weighted sum of the individual
computations in the ratio 60:40. Higher weightage is given to fuzzy computed value as it
estimates the tuning parameter slightly on a lower side.

6.3.1 Tuning moderation

The estimated values of the tuning parameters are aggregated by assigning suitable
weights to the outputs generated by both Neural and Fuzzy subcomponents for each of the
tuning parameters. These estimated values may sometimes be ineffective or if overestimated may result in system instability and hence, certain moderation technique must be evolved to ensure effective use of the tuning parameters in improving the query response-time. In this section, we present a novel moderation technique based on the relative impact on the tuning parameters described as below: The impact factor of a tuning parameter $T_p[i]$ for a given Workload Type $j$ is defined as:

$$IFactor(T_p[i], j) = \frac{\frac{\delta R_{time}}{\delta T_p[i, j]}}{\sum_{i=0}^{TPN} \frac{\delta R_{time}}{\delta T_p[i, j]}} \ldots \ldots \ldots \ldots \ldots \ldots \ldots (5.1)$$

Where, $TPN$ is the number of dynamically tunable parameters. $\delta R_{time}$ is change in query response-time with respect to $\delta T_p[i]$ is the change in $i^{th}$ tuning parameter for a given workload type $j$. Basically, this formula is based on the slopes at each value of the tuning parameters in the characterization graphs of Response-time v/s tuning parameters. The following module shows the tuning moderation.

```plaintext
//Moderate the estimated values based on the impact factor of each tuning parameter
for i ← 1 to TPN  do  //For each Tuning Parameter
    IFactor(Tp[i], wType) = \left( \frac{\Delta R_{time}}{\Delta Tp[i, wType]} \right) / \left( \sum_{i=1}^{TPN} \frac{\Delta R_{time}}{\Delta Tp[i, wType]} \right)  //Compute tuning impact
    Tp[i]=IFactor(Tp[i], j)*Tp[i]
    performTuning(Tp[i])                                  //Perform tuning action
next i
```

As can be observed from above, the tuning moderation involves extra computations. Further, it is difficult to find the slopes on the graphs as they are not continuous and in other types of workloads for example, TPC-H, some response-time graphs do not have slopes to compute. Hence, this method is not applicable to all workload types. For these two reasons, tuning moderation does not offer any advantage apart from limiting the tuning parameter values within limits. The moderation step can easily be incorporated in the training data-set and hence, this step is not included in the algorithm listed above.
6.4 Result Analysis of Neuro-Fuzzy tuning

The method is applied to all the workload scenarios including TPC-C, D, E and H and also scaling factors 2, 5 and 10 and the results show significant improvement in performance over the Auto-tuning feature of Oracle 10g.

![Figure 6.4a. Performance comparison of Neuro-fuzzy approach (Current Run) with Auto-tuning feature of the DBMS, (SF=2) (28% improvement)](image)

The performance improvement varies from workload to workload and also within a given workload type, the performance depends on the scaling factor (Size of the database). Figure 6.4a shows a performance improvement of 28% for TPC-C workload with a scaling factor of 2.

![Figure 6.4b. Performance comparison of Neuro-fuzzy approach (Current Run) with Auto-tuning (Run20) feature of the DBMS, (SF=5) (6.5% improvement)](image)

Figure 6.4b shows a performance improvement of 6.5% for the same workload type but with a scaling factor of 5. The usage of system memory for tuning is efficient by 20% for 50 user load and 22.99% on an average over the user-load range of 5-50 users as shown in the memory usage bar graph in figure 6.5. This saving in memory is because of the Neuro-fuzzy’s conservative way of gradually increasing the memory based on current user-load, workload
type etc. On the contrary the built-in self-tuning approach uses over-provisioning anticipating possible increase in user-load or change in workload type.

![Memory Usage Chart 10X - TPC-C Workload](image)

**Figure 6.5. Memory Usage : Neuro-Fuzzy V/s Auto-Tuning**

### 6.4.1 Performance improvement in other workload types

The results shown in figure 6.6a-6.6d correspond to Workload types TPC-E, TPC-H and TPC-D respectively. As can be seen from the graphs the performance improvement is higher as compared to workload analysis approach used in Oracle 10g. This can be ascribed to the fact that all the tuning parameters are effectively used for performance improvement by the proposed self-tuning technique.

![Performance comparison of Neuro-fuzzy approach(Current Run) with Auto-tuning feature of the DBMS.(SF=2, TPC-E Workload) (5.76%)](image)

**Figure 6.6a. Performance comparison of Neuro-fuzzy approach(Current Run) with Auto-tuning feature of the DBMS.(SF=2, TPC-E Workload) (5.76%)**

The performance improvement for a scaling factor of 2 for TPC-E workload is 5.76% whereas for the same workload and for a scaling factor of 1, the performance improvement was almost same as that of Neural which is 32.8%. This shows that the database size has a very significant impact on performance.
These results obtained clearly indicate that the Neuro-fuzzy controlled tuning has slightly better performance at lower scaling factor as tuning technique has enough headroom to change the tuning parameters to the most effective values. However, at higher scaling factors the performance graph coincides with the Auto-tuning feature provided by Oracle 10g as, at higher scaling factor there is not enough room for further change in the allocated memory as the combination of all the tuning subcomponents would have reached the SGA_TARGET value.

The performance improvement for TPC-H workload(fig. 6.6b) is 11.51% and that for TPC-D workload(fig. 6.6c) is 25.93%. TPC-H being heavier workload as compared to TPC-D the performance improvement is only 11.51%.
workload conditions. The performance of the Neuro-Fuzzy approach varies from workload to workload and the performance improvement as compared to the Auto-tuning feature of Oracle10g was found to be in the range of 0-28% and also a 20% saving in memory usage for TPC-C workload type. On the other hand the performance improvement in TPC-H was only 11.51% and that for TPC-D 25.9%. This improvement was achieved through efficient and effective use of the dynamic tuning parameters by the adaptive techniques proposed as part of this work. It is also observed that combining Neural and Fuzzy approaches doesn’t provide any performance advantage. However, the response-time is pretty flat over the entire userload range. However, further research is needed to validate the working of the proposed self-tuning techniques on other popular databases like MySQL, MSSqlServer, PostgresSQL etc. to prove that these techniques are generic. Further work needs to be carried out to find out at what interval and under what conditions triggering the tuning action has the most predictable response-time.

6.4.2 Throughput comparison

As shown in figure 6.6d, under the TPC-E workload-type with scaling factor of 1, an improvement of 6.6% in throughput was observed as compared to auto-tuning feature of Oracle 10g. This improvement performance is mainly due to efficient usage of shared memory that enables execution of more queries in a given amount of time. However, under TPC-C workload type, no improvement in throughput was observed. This may be because of smaller number of record manipulations in OLTP workload-type as compared to DSS workload-type.

![Figure 6.6d Throughput comparison under TPC-E Workload](image)
6.5 Performance tuning in a distributed database environment

The distributed database systems [16][108] provide some of the important benefits like scalability, reliability and availability etc. However, due to non-availability of expert DBA at all the database locations, some of the database systems may be ill-tuned, resulting in overall degradation in performance. It is desirable to have a centralized self-tuning system that monitors the performance of each of the database instances running on remote servers and carryout the necessary tuning action to bring up the performance level of the applications running on these distributed databases [71].

A typical distributed application scenario would be as shown in figure 6.7, wherein each individual server would be supporting applications of certain type. Since, the workload generation tool does not support facility to distribute the workload on different database servers, a distributed scenario is simulated by querying the database spread over three database server instances. Each of these server instances are loaded with data of TPC-C workload-type with Scaling Factor set to 1 and will receive the queries from the workload generation tool.

The client application prepares a set of SQL statements representing an operation of an order processing system and sends the same to one of the database instances for execution. The database instance, checks for the data locally and if the entire data is not found on that local server, it would query the other database instance for the remaining data and the combined result would be posted back to the client.

Figure 6.7 Distributed database environment

The client application prepares a set of SQL statements representing an operation of an order processing system and sends the same to one of the database instances for execution. The database instance, checks for the data locally and if the entire data is not found on that local server, it would query the other database instance for the remaining data and the combined result would be posted back to the client.
6.5.1 Validation of proposed self-tuning technique

The experimental setup shown in figure 6.7 was used to validate the results. Three set of experiments were conducted and the results are plotted. As can be seen from figure 6.8, the response-time of the system for without tuning (with default settings) sharply rises as the number of users are increased beyond 75. The performance of the system is little better for system with auto-tuning on. The performance improvement is better only at higher userload as compared to without tuning case. However, as can be seen from the graphs, the self-tuning with Neural network approach has significantly lower response-time throughout the user range of 10-80. The performance improvement is 18.64% in NN tuning as compared to Auto-tuning. The distributed database experiment was setup on three desktop PCs with 2.8GHz CPU, 4GB RAM and 80GB Hard-disk. Deploying heavier load on systems with less powerful PC and lower RAM sizes was causing system instability and hence, testing of the proposed technique with heavier workload types (TPC-H, TPC-E etc.) with higher scaling factors of TPC-C could not be carried out.

Summary

The objective of implementing Neuro-fuzzy approach was to combine the best of both the approaches. The performance improvement in terms of response-time was marginal, an almost flat response for TPC-C workload type of SF=5 was observed. This is very important in a critical business environment where predictable response-time is most desirable than smaller response-time. The Neuro-fuzzy based tuning architecture presented in this chapter has been tested using all the four workload types and the results obtained show a significant improvement across all the
workload-types, user-loads and scaling factors. The performance varies from workload to
workload and furthermore, the performance improvement achieved varies within a given
workload depending on the scaling factor. The improved performance using all the three self-
tuning techniques can be ascribed to effective utilization of the tuning parameters and accurate
estimation of tuning parameters which is based on experimental data input. Further, the light-
weight computational step in extracting input parameters as compared to exhaustive analysis of
data by the traditional approach was also one of the important reasons for improved
performance. The proposed method has been validated in a distributed database environment
and found to be better by 18.64% as compared to workload analysis approach of Oracle 10g
DBMS.