ADMiRe: An Algebraic Approach to System Performance Analysis Using Data Mining Techniques

Kien A. Hua
University of Central Florida
kienhua@cs.ucf.edu

Ning Jiang
University of Central Florida
njiang@cs.ucf.edu

Roy Villafane
University of Central Florida
villafan@cs.ucf.edu

Duc Tran
University of Central Florida
dtran@cs.ucf.edu

ABSTRACT

System performance analysis is a very difficult problem. Traditional tools rely on manual operations to analyze data. Consequently, determining which system resources to examine is often a lengthy process, where many problems are elusive, even when using data mining tools. We address this problem by introducing the Analyzer for Data Mining Results (ADMiRe) technique as a natural and flexible means to further interpret data mining outcome. In our scheme, regression analysis is first applied to performance data to discover correlations between parameters. Regression rules are defined to represent this output in a format suitable for ADMiRe. ADMiRe expressions are then used to manipulate these sets of rules, revealing information about combined, common and different features of varying configurations. This knowledge would be unavailable if regression output were considered in isolation. ADMiRe was tested with performance data collected from a TPC-C (Transaction Processing Performance Council) test on an Oracle database system, under various configurations, to demonstrate the effectiveness of our technique.

General Terms

Algorithms, Measurement, Performance.

Keywords

Scalability, Data Mining, Regression.

1. INTRODUCTION

Solving system performance problems [3] [9] [13] [14] [15] is becoming increasingly crucial due to the explosive growth in the scale and complexity of information systems. Many organizations provide online transaction processing and other services, so good performance of their computing systems is vital to their existence. When performance problems arise, many users try to solve them by purchasing more advanced hardware, when a better solution would involve revamping some of the system's logic.

To precisely discover the cause of system degradation, performance data on different sub-components of a system is exploitable. There are many utilities that capture this data. Their output is usually a time series. Determining which system resources to examine is often a lengthy process of educated guesswork, where many problems can run undetected. However, few attempts have been made to capture the similarities and differences under various system configurations. In [9], the authors show that self-similar ([4] [11]) behaviors exist in high-level file system events. However, in the paper, they only focus on the file system and only very few attributes are examined. [3] [15] [13] [14] [10] [17] and numerous other papers also deal with performance issues, but are either focused on a specific subsystem such as I/O or network or aimed at a specific software system such as Oracle DBMS or the Sun Solaris Operating system. More focus is required for techniques that can be applied to generic system performance analysis.

To address these problems, we propose the *Analyzer for Data Mining Results* (ADMiRe) technique, which uses results from data mining ([2] [5] [7] [6] [8] [12] [16] [19]) operations on system performance data. We argue that data mining techniques are very useful for system performance analysis due to the completeness of the results. This systematic approach allows us to thoroughly examine the relationships among large number of parameters of different subsystems. We have implemented a system based on the ADMiRe technique. Our methodology consists of three steps:

Rule Extraction: Performance data for different system configurations is collected by various tools. We apply regression analysis to summarize each configuration's data into a set of *regression rules*. This rule set captures the behavior of the system in a highly compact form.

Rule Analysis: We manipulate these rule sets by applying operators to reveal similarities, differences, and other information among different system states represented by the various rule sets.

Result Ranking: The information is subsequently ranked according to certain quantitative means to help the user focus on the more likely causes of the performance problems.

The contributions of this paper are as follows:

1) Propose using data mining as a systematic approach to address the high complexity of system performance analysis problems.

2) Define regression rules as a means for summarizing performance data in a very compact form. To the best of our knowledge, this concept has not been explored in the literature. It offers an effective way to analyze very complex performance problems.

SAC 2003, Melbourne, Florida, USA

3) Introduce an algebraic approach to analyzing system performance. The user can uncover causal effect by manipulating regression rules using rule algebra.

Although the use of data mining allows us to cover a broader span of subsystems, we do not claim that other techniques are less important; depth of analysis is as important as breadth. The benefits of ADMiRe reach beyond system performance evaluation. This method can also be used to study many other topics such as market analysis, human behavior, etc.

The rest of the paper is organized as follows. In Section 2, we first give a motivating example and then formally define our research problem. We define *regression rules*, describe the rule operators and introduce the ranking of regression rules in Section 3. The implementation and complexity analysis of the algorithms are presented in Section 4. Section 5 provides experimental results on an Oracle database. Concluding remarks are given in Section 6.

2. PROBLEM DEFINITION AND MOTIVATION

The aim of this study is to provide useful information for tuning a computer system. Consider a database system. It consists of various hardware and software components (e.g. communication network, disks, CPU, memory, lock manager, log manager, buffer manager, etc.). Understanding dependencies between these components is imperative for tuning. We might observe a correlation between disk access rates and network usage. This knowledge could encourage the system administrator to relocate data to another server. However, further gains might be achieved by comparing the sets of correlations observed at various system states. If performance degrades after adding processors, comparing the subsystem performance correlations before and after the upgrade may reveal the cause of the problem. The existence or absence of particular correlations may explain certain system behavior. However, such information is hard to uncover if we consider the system states in isolation. As far as we know, no research has been done in this direction.

System performance data collected by many tools generate output in the form of time series. Parameter values corresponding to a measurement of some system property (i.e., CPU busy time) are generally measured once at each of equally spaced time intervals. Formally, a time series can be defined as follows:

$A = \{a_1, a_2, \ldots, a_n\}$, $1 \leq j \leq n$, is a set of system parameters.

$TS = \{obs_1, obs_2, \ldots, obs_p\}$ is a time series, where each

$obs_i = <t, v_{1i}, v_{2i}, \ldots, v_{ni}>$ is a tuple corresponding to measurements with timestamp t and value v_{ki} of each a_k.

We focus our analysis on a set of several time series, $T = \{s_1, s_2, \ldots, s_m\}$. Each s_i corresponds to performance data captured under some particular system state. The commonalities are extracted and differences from the output of data mining analysis. The final results are ranked based on certain importance metrics.

3. KNOWLEDGE DISCOVERY FROM MULTIPLE REGRESSION ANALYSIS RESULTS

We apply linear regression analysis ([16]) to investigate the set T of time series, in order to discover relationships among different parameters within each individual s_i. It is applied to each pair of parameters P_1 and P_2 from a time series. The existence or absence of a correlation in one data series and not another is useful for explaining certain behaviors. In order to obtain such information, we employ three operations: Union ($\cup$), Intersection ($\cap$) and Subtraction ($-$). Union is used to combine sets of correlations. Intersection reveals common correlations among different regression analyses. Subtraction can reveal behavior exclusive to one configuration. Knowledge discovery of similarity and difference characteristics among such results is accomplished by constructing algebraic expressions with these operators.

3.1. Regression Rules

Regression rules represent the augmented output of regression analysis. These are necessary to facilitate further manipulation of sets of correlations generated by regression analysis. A *regression rule* is a tuple:

$$<P_1, P_2, RSS>, P_1, P_2 \in A$$

Where *RSS* is a *set* of *regression structures*. Each regression structure $rs \in RSS$ has these members:

rr[*x,y*]: An interval that specifies a *range* of coefficients of correlation r^2 of attributes P_1 and P_2, having $x<y$, referred to as the "*r-range*".

mbs: A set of $<m, b>$ pairs associated with an *r-range*, where m is the slope of the line and b is the y-intercept.

A coefficient of regression is a numeric point value. However, we represent regression rules using an interval [*x,y*], having $x \leq y$, for the coefficient of regression. This implies existence of a *range* of coefficient of regressions for a pair of attributes. Assuming that attributes P_1 and P_2 have a linear relationship with r^2=0.85, suppose a regression rule R with interval [0.0,0.85] is constructed. The implication is valid at 0.85, because the coefficient originally states this fact. The implication is still valid at a value of r^2 of 0.6, for example. Afforded some flexibility, if we were to pose the query "list all attributes pairs which satisfy a correlation strength of 0.6", the answer would be "yes" for P_1 and P_2 because a pair of attributes whose values are correlated by a regression relationship having an r^2 statistic of 0.85 exceed, and thus satisfy, the requirement of the query. If the query had specified a value of r^2 of 0.9, then this attribute pair would be excluded because it lacks the required relationship strength. Thus initially, from a correlation relationship of strength r^2, a regression rule is constructed with a range of $[0,r^2]$. A generic correlation interval [*x,y*] can then be viewed as the result of posing a conjunctive query on many regression rules. Given a regression rule $R_1=<P_1,P_2,RSS>$, *regression rule* $R2=<P_1',P_2',RSS'>$ is a corresponding *rule* of R_1 if $P_1 = P_1'$ AND $P_2 = P_2'$.

We also define an order for intervals. For any two intervals $rr_1[a,b]$ and $rr_2[c,d]$, $rr_1 \prec rr_2 \Leftrightarrow a<c$. *r-ranges* are stored in ascending order in each *RSS*. In addition, all the *r-ranges* in an *RSS* have disjoint intervals (i.e., for any two ranges $rr_1[a,b]$ and $rr_2[c,d]$, either $b<c$ *or* $d<a$). The monotonicity and disjointness requirements are crucial for good performance of the operators. Additional useful properties and functions are defined as follows:

$|RSS|$ = number of regression structures in the regression structure set *RSS*.

$|rs|$ = number of <*m, b*> pairs in the regression structure *rs*.

rs.rr is the rr range of a regression structure *rs*.

mbp(rs,i) is the ith <*m,b*> pair of regression structure *rs* given the $\prec$ relation.

$sim(<m_1,b_1>,<m_2,b_2>) = <mfactor,bfactor>$,

$$mfactor = \begin{cases} 1 & \text{if } m_1 \text{ and } m_2 \text{ are of different signs.} \\ 0 & \text{if } m_1 = m_2 = 0. \\ (\max(|m_1|,|m_2|) - \min(|m_1|,|m_2|)) / \max(|m_1|,|m_2|) & \text{otherwise.} \end{cases}$$

$$bfactor = \begin{cases} 1 & \text{if } b_1 \text{ and } b_2 \text{ are of different signs.} \\ 0 & \text{if } b_1 = b_2 = 0. \\ (\max(|b_1|,|b_2|) - \min(|b_1|,|b_2|)) / \max(|b_1|,|b_2|) & \text{otherwise.} \end{cases}$$

The *sim* is a similarity measurement of a pair of <*m,b*> pairs. Similarity is defined in terms of the absolute changing rate of the *m* and *b* values. *mfactor(bfactor)*=0 implies no changes; *mfactor(bfactor*)=1 implies maximum change. In practice, two user defined thresholds MFACTOR_THRESHOLD and BFACTOR_THRESHOLD are applied to decide the similarity of two <*m, b*> pairs.

Operators between two sets of <m, b> pairs are defined as follows:

$$\{mbs_1 \cup mbs_2 = \{<m,b> \mid <m,b> \in mbs_1 \text{ OR } <m,b> \in mbs_2\}\}$$

$$\left\{\begin{array}{l} mbs_1 \cap mbs_2 = \{<m,b> \mid (<m,b> \in mbs_1 \text{ AND } \exists <m_1,b_1> \in mbs_2 \\ (sim(<m,b>,<m_1,b_1>) < \{<MFACTOR\_THRESHOLD, BFACTOR\_THRESHOLD>\}) \\ \text{OR } (<m,b> \in mbs_2 \text{ AND } \exists <m_1,b_1> \in mbs_1 \\ (sim(<m,b>,<m_1,b_1>) < \{<MFACTOR\_THRESHOLD, BFACTOR\_THRESHOLD>\})\} \end{array}\right\}$$

$$\left\{\begin{array}{l} mbs_1 - mbs_2 = \{<m,b> \mid (<m,b> \in mbs_1 \text{ AND} \neg \exists <m_1,b_1> \in mbs_2 \\ (sim(<m,b>,<m_1,b_1>) < \{<MFACTOR\_THRESHOLD, BFACTOR\_THRESHOLD>\})\} \end{array}\right\}$$

3.2. Operations On Regression Rules

Before we can define operators for regression rules, operations on regression structures are needed. In a regression rule, the r-range represents the strength of a given relationship between system attributes. There are 13 relationships for r-range intervals [1], many being symmetric. In ADMiRe, two functions are defined to obtain the overlapped and non-overlapped parts of a pair of intervals.

$$rs_1.rr \cap rs_2.rr = \{s \mid s \in rs_1.rr \text{ AND } s \in rs_2.rr\}$$

$$rs_1.rr - rs_2.rr = \{s \mid (s \in rs_1.rr \text{ AND } s \notin rs_2.rr)\}.$$

In this context, we define binary operators for a pair of *regression structures* rs_1 and rs_2 as follows

Union (∪):

$$rs_{\text{union1}}.rr = rs_1.rr \cap rs_2.rr;\ rs_{\text{union1}}.mbs = rs_1.mbs \cup rs_2.mbs.$$

$$rs_{union2}.rr = rs_1.rr - rs_2.rr;\ rs_{\text{union2}}.mbs = rs_1.mbs.$$

$$rs_{union3}.rr = rs_2.rr - rs_1.rr;\ rs_{\text{union3}}.mbs = rs_2.mbs.$$

Intersection (∩):

$$rs_{\text{inter}}.rr = rs_1.rr \cap rs_2.rr;\ rs_{\text{inter}}.mbs = rs_1.mbs \cap rs_2.mbs$$

Subtraction (–):

$$rs_{\text{sub1}}.rr = rs_1.rr \cap rs_2.rr;\ rs_{\text{sub1}}.mbs = rs_1.mbs - rs_2.mbs.$$

$$rs_{sub2}.rr = rs_1.rr - rs_2.rr;\ rs_{\text{sub2}}.mbs = rs_1.mbs.$$

In the definition of the operators, the *r-ranges* are operated in combination with the *m* and *b* coefficients. The similarity and difference are defined in terms of all the regression coefficients. Consider the intersection operator. If two source regression structures have common correlation strength (overlapping *r-ranges*), the overlapping *r-range* is common only if the respective *m* and *b* coefficients are similar as formerly defined. The operations of the union and subtraction operator are likewise derived.

Now let us consider *regression rules*. Although we present operators between two sets of regression rules RS_1 and RS_2, ADMiRe can process arbitrary algebraic expressions consisting of multiple operators.

Sometimes regression rule sets of similar system configurations should be combined for comparison to other rule sets using the Union (∪) operator. It encapsulates the combined information content for both rule sets. For any two corresponding regression rules $R_1 \in RS_1$ and $R_2 \in RS_2$, a single destination rule is created. We apply this operator on each such pair of regression structures and insert the subsequently generated structures into the destination rule in *r-range* order. Rules in one set having no counterpart are incorporated verbatim.

Intersection (∩) as an operator selects the regression rules common to both rule sets. It captures similar characteristics. Any rule present only in a single set is excluded. For any two corresponding rules a single destination rule is generated. Intersection is performed on each pair of regression structures of the two rules. Resulting regression structures are also inserted into the target rule in ascending order of their *r-ranges*. Rules in RS_1 and RS_2 that have no *corresponding rules* in the other set are simply discarded.

The Subtraction (–) operator in $RS_1 – RS_2$ outputs all the regression rules that are found only in set RS_1 but do not exist in RS_2. Application of the subtraction operator on sets of regression rules is more complex than the other operations described so far. The main idea is to eliminate the similarity among correlations. Again, there are two situations. For one regression rule $R_1 \in RS_1$, consider its *corresponding rule* $R_2 \in RS_2$, a single destination rule is generated. Each regression structure $rs \in R_1$ is iteratively subtracted by regression structures in R_2 that have *r-ranges* overlapping with *rs.rr*. The subtracted results are inserted into the destination rule. By storing the *r- ranges* in a *RSS* in ascending order the retrieval of overlapping *r-ranges*, the iterative operations on those ranges and the insertion of resulting regression structures can be performed in linear time with respect to the number of regression structures. Rules in RS_1 that have no *corresponding rules* in RS_2 are directly inserted into the resulting rule set.

Knowledge about the commonalities and differences between various data sets is extracted by creating expressions in algebraic like format. Experimental results (Section 5) demonstrate that the

ADMiRe scheme is very flexible and can uncover important statistics to improve system performance.

3.3. Ranking of ADMiRe Output

Manipulation of different sets of regression rules can yield important information for the user to understand system behavior. However, the output of ADMiRe is usually very large. It is a daunting task for the user to make the right judgment of which parameters to tune from such a vast number of resultant regression rules. Hence, it is imperative to rank regression rules based on certain importance measurement to help the user focus on rules that are most critical to system performance.

In ADMiRe, a regression rule is usually associated with a set of regression structures. Each regression structure has a *r-range* and several <*m,b*> pairs. Theoretically, all the regression coefficients should be considered to determine the importance of a regression rule. However, in practice, it is too time consuming to compare each pair of regression structures of two regression rules. A more efficient approach needs to be designed.

We decide to calculate the rank of a regression rule *R* by comparing the maximum interval of *r-ranges* among all its regression structures against the counterparts of other regression rules. We define the maximum interval of a set of *r-ranges* as follows:

$$Max(rr_1[a1, b_1], rr_2[a_2, b_2], \ldots, rr_n[a_n, b_n]) =$$

$$rr_k \text{ such that } rr_k \in \left\{ rr_i[a_i, b_i] \middle| \forall rr_j[a_j, b_j] (b_i - a_i \geq b_j - a_j) \right\}$$

This should not be confused with the partial order $\prec$ defined on *r-ranges*. In ADMiRe, the larger the interval that the maximum *r-range* of a regression rule spans, the more important the particular rule. This is valid for all the operators. First, consider the resultant regression rules of the Union operator. A larger resultant *r-range* implies that the union of correlations between two parameters covers a broader range in terms of regression strength and is more important than other regression rules. Second, consider the resultant regression rules of the Intersection operator. The purpose of the Intersection operator is to discover common behaviors within two system states. User can derive from a regression rule with a larger maximum *r-range* interval that the two source rules have more overlapping in their respective *r-ranges* and are thus more similar to each other. Therefore, the resultant regression rules deserve more investigation. Third, for the Subtraction operator, a larger maximum *r-range* interval means that one of the two source corresponding rules has a larger *r-range* span (thus stronger regression strength) than the other. Hence, system behavior in terms of the two parameters is more diversified and the resultant regression rules are of more interest to the user.

In ADMiRe, after a user issued expression is calculated, the output set of regression rules is sorted in descending order of the importance of the rules. User can focus on the most important correlations discovered by the system.

4. SYSTEM IMPLEMENTATION

Operators are defined on two *corresponding regression rules*. The pseudo code of the operators is presented in this section.

Subtraction (R_1- R_2):

1. Create a new regression rule R_{sub};
2. $R_{sub}.P_1 = R_1.P_1$;
3. $R_{sub}.P_2 = R_1.P_2$;
4. for {each rs_i in the $R_1.RSS$} do
5. $temp\_rs = rs_i$;
   RSS_0={rs_0 | $rs_0 \in R_2.RSS$ AND $rs_0.rr$ overlaps, contains or is contained by $rs_i.rr$ };
6. if $RSS_0 \neq \Phi$ then
7. while $temp\_range.rr \neq \phi$ do
8. for {each rs_k ($rs_k \in RSS_0$)} do
9. $temp\_range = temp\_range - rs_k$;
10. Insert *temp_range* to $R_{sub}.RSS$;
11. end for;
12. end while;
13. else
14. Insert rs_i to $R_{sub}.RSS$;
15. end if;
16. end for;

Due to the sorted and disjoint property, for each rs_i in R_1, the time complexity for generating the RSS_0 only requires starting from the last regression structure in the RSS_0 of rs_{i-1}. This makes the processing time linear. Finding the similar <*m,b*> pairs, however, requires repeated scanning. The worst-case time complexity for the subtraction operator is

$$\left(\left(|R_1.RSS| + |R_2.RSS| \right) + \left(\sum_{rs_j \in RS_1.RSS} |rs_i| \right) \cdot \left(\sum_{rs_j \in RS_2.RSS} |rs_j| \right) \right).$$

For two regression rule sets RS_1 and RS_2, to calculate RS_1- RS_2, let $|RS_1|=n$, $|RS_2|=m$. We first sort RS_2 according to its P_1 and P_2, in time $O(m \log m)$. For each regression rule $R_1 \in RS_1$, we search for a regression rule in RS_2 having the same P_1 and P_2 as R_1. Using binary search results in $O(\log m)$ complexity. Overall, the worst case time complexity to subtract two regression rule sets is

$$\left(m \log m + n \cdot \left(\log m + \left(\left| R_1.RSS \right| + \left| R_2.RSS \right| \right) + \left(\sum_{rs_i \in RS_1.RSS} |rs_i| \right) \cdot \left(\sum_{rs_j \in RS_2.RSS} |rs_j| \right) \right) \right).$$

In practical situations, $|RSS|$ and $|rs|$ are below 10, and parameters m and n are dominant. The time complexity for $RS_1 - RS_2$ is $O((m+n) \log m)$. The sorted and disjoint properties greatly reduce the hidden factor

(2) Intersection ($R_1 \cap R_2$):

1. Create a new regression rule R_{int};
2. $R_{int}.P_1 = R_1.P_1$;
3. $R_{int}.P_2 = R_1.P_2$;
4. for {each rs_i in $R_1.RSS$ } do
5. Generate the set of regression structure RSS_0={rs_0 | $rs_0 \in R_2.RSS$ AND $rs_0.rr$ overlaps, contains or is contained by $rs_1.rr$}.
6. if $RSS_0 \neq \Phi$ then
7. for {each ($rs_k \in RSS_0$) } do
8. insert $rs_i \cap rs_k$ into R_{int}.RSS.
9. end for;
10. end if;
11. end for;

The worst-case time complexity of the Intersection operator is same as the Subtraction operator. It is also $\left(m\log m + n\cdot\left(\log m + \left(\left|R_1.RSS\right| + \left|R_2.RSS\right|\right) + \left(\sum_{rs_i \in RS_1.RSS}\left|rs_i\right|\right)\cdot\left(\sum_{rs_j \in RS_2.RSS}\left|rs_j\right|\right)\right)\right)$. Similarly, the sorted and disjoint properties greatly reduce the hidden factor.

(3) Union ($R_1 \cup R_2$):

1. Create a new regression rule R_{union};
2. $R_{union}.P_1 = R_1.P_1$;
3. $R_{union}.P_2 = R_1.P_2$;
4. for {each rs_i in $R_1.RSS$ } do
5. Generate the set of regression structure RSS_0={rs_0 | $rs_0 \in R_2.RSS$ AND $rs_0.rr$ overlaps, contains or is contained by $rs_1.rr$}.
6. for {each ($rs_k \in RSS_0$) } do
7. insert $rs_i \cup rs_k$ into R_{int}.RSS*.
8. end for;
9. end for;
10. Insert all the remaining regression structures in R_2 into $R_{union}.RSS$.

*When inserting union results into R_{union}. *RSS*, the final list should meet the ascending and disjoint requirement.

The union of two regression rules, R_1 and R_2, however, involves duplication reduction and order and disjoint preservation. We have to examine each pair of <*m, b*> pairs in step 7. Emphasis must be put on combining the regression structures. Since the final *RSS* should have the sorted and disjoint property, step 10 involves a binary search of the intermediate results, which takes *O(log p)* time (*p* is the length of the intermediate *RSS*). Like before, the scanning of <*m, b*> pairs is dominant. The worst case time complexity for combining two rules is

$$\left(\left(\left|R_1.RSS\right| + \left|R_2.RSS\right|\right) + \left(\sum_{rs_i \in RS_1.RSS}\left|rs_i\right|\right)\cdot\left(\sum_{rs_j \in RS_2.RSS}\left|rs_j\right|\right) + \left|R_1.RSS\right|\cdot\log\left|R_2.RSS\right|\right).$$

To perform a union of two regression rule sets, RS_1 and RS_2, according to the definition, all the rules in both RS_1 and RS_2 should be included in the result set. In other words, we need to scan both of the rule sets to generate the final result. The worst-case time complexity is

$$\left(m\log m + n\cdot\left(\log m + \left(\left|R_1.RSS\right| + \left|R_2.RSS\right|\right) + \left(\sum_{rs_i \in RS_1.RSS}\left|rs_i\right|\right)\cdot\left(\sum_{rs_j \in RS_2.RSS}\left|rs_j\right|\right) + \left|R_1.RSS\right|\cdot\log\left|R_2.RSS\right|\right) + m\right).$$

In practical situations, |*RSS*| and |*rs*| are less than 10, with parameters m and n being dominant. The time complexity for $RS_1 \cup RS_2$ is also *O((m+n) log m)*.

5. EXPERIMENTAL STUDY

To validate our approach, we investigated an Oracle database management system running the TPC-C Benchmark and a web server. Different workloads are generated for both cases under various CPU and I/O configurations. ADMiRe was used to analyze the systems.

The Transaction Processing Performance Council's TPC-C [18] benchmark suite was run on an Oracle 8i DBMS. This benchmark is an industry standard for evaluating the OLTP (On-Line Transaction Processing) performance of a DBMS. It ran on a Sun Enterprise 450 Server with 4 250Mhz UltraSparc processors, 2 GB of RAM, and 48 GB of storage striped across 12 disks. System load and capacity were systematically varied as shown in Table 5.1. Hereafter, we use "system *s*" to refer to the system configuration that data set *s* pertained to.

Table 5.1. System Configurations and Data sets

Dataset	User Factor	Processors
1	5	4
2	10	4
3	20	4
4	30	4
5	30	2

In each configuration, the system was allowed to reach a steady state. Operating system and database performance data was collected at 10-second intervals for a total of 600 seconds. The system scaled well on scenarios 1, 2, and 3. In the 4th and 5th scenarios, the system reached an overloaded condition, where users are not served in the expected time

We consider two sample expressions here. In showing resulting regression rules, we will use the following notation :

P1: (name of first attribute),

P2: (name of second attribute)

RSS: {<[*r2Low*,r2High]:<m_1,b_1>,<m_2,b_2>, ...>, ... }, where [*r2Low,r2High*] indicates the r^2 interval for this rule, and following each such interval is a set of associated <*m,b*> value pairs

Table 5.2. Output of expression (5-4)

P1	*P2*	*RSS*
latch_free: total_timeouts	*Sml_mem*	{ <[0.311859,0.824123]: <51.6891,33582400>> }
process_alloc: immediate_gets	*Sml_mem*	{ <[0.330459,0.82736]: <3050.08,32718200>> }
file_open: total_waits	*Sml_mem*	{ <[0.302001,0.780548]: <205.18,33014300>> }
TPC_DATA1: pbw	*Sml_mem*	{ <[0.192184,0.773805]: <107.218,32428500>> }
pmon_timer: total_waits	*Sml_mem*	{ <[0.283592,0.763297]: <744.243,30635500>> }
TPC_DATA1: pbr	*Sml_mem*	{ <[0.231663,0.75585] <109.285,32234300>> }

Consider expression (5-4). Some relevant top-ranking rules are shown in Table 5.2. The word description of the (5-4) test is the "regression relationships that are present in a 30-user & 2-CPU run which are not present in a 30-user 4-CPU run". For example, the first regression rule indicates that there was a linear correlation among the *latch_free:total_timeouts* database parameter and the *sml_mem* allocations operating system parameter. Looking at the 2-CPU case (system 5), the coefficient of correlation (r^2) was 0.824123. Since this regression rule was constructed directly from regression analysis data, the associated range is [0.0,0.824123]. In the 4-CPU case (system 4), the correlation of these parameters was 0.311859, with a

corresponding *regression rule* range of [0.0,0.311859]. Application of the discussed operators yields the rule with range [0.311859,0.824123]; this fares well with the definition of the classical subtraction operation in which the remaining portion does not contain any portion from the subtracted argument.

The important rules are relevant to the *sml_mem* parameter, which is an operating system parameter that reflects the number of small memory allocation requests (small requests are less than 256 bytes). The correlation between the *sml_mem* parameter with various database internal parameters indicates a bottleneck in operating system virtual memory pool when system 5 was experiencing a high workload. It was busy servicing small memory requests incurred by various database requests such as process and tablespace memory allocation. For example, consider the rule *<TPC_DATA1:pbw,sml_mem>* and *<TPC_DATA1:pbr,sml_mem>*. *TPC_DATA1:pbw* captures the behavior of write operations on the *TPC_DATA1* tablespace of the database, and the *pbr* variant is the same for read operations; the data for the TPC-C tables was stored in these tablespaces. From a system point of view, one might deduce that the stronger correlation between tablespace I/O operations and small memory allocations might be a sign of the database relying more on the operating system due to an overload situation. In contrast, such overload did not exist in system 4. The system was still able to deal with an increasing number of small memory requests. Operations that would have otherwise involved the operating system were done internally within the DBMS; such operations were thereby shielded from the operating system and its performance statistic counters. Consequently, the respective rule appeared in the result of expression (5-4). More importantly, these rules also indicate how the performance of system 5 can be improved. Obviously, a naïve approach is to add some memory to the system

Table 5.3. Output of expression (4-1)–(2-1)

P_1	P_2	*RSS*
Active checkpt queue Latch: immediate misses	*SQL*Net message from client: total_waits*	{ <[0.014638,0.706388]: <0.018000,45.000000>>}
CPU1:intr	*CPU2:intr*	{ <[0.212138,0.651438]: <0.925367,15.386000>>}
CPU0:idl	*File-sz*	{ <[0.075037,0.758398] :<-4.38818,1614.34>>}
Freemem	*lg_memalloc*	{ <[0.206937,0.77661]: <-51.0073, 8266>>}

Now let us consider the expression (4-1)-(2-1) as the second example. The operations (2-1) and (4-1) both attempt to capture "relationships which remain when the system load is increased". Consequently, the more complex expression (4-1)–(2-1) identifies "relationships which remain when a system load increase results in an overload condition". Table 5.3 lists a few of top-ranking rules out of around 3500 resultant rules.

Consider the rule involving parameters "*freemem*" and "*lg_memalloc*". The "*freemem*" parameter records the amount of available user memory of the operating system. The "*lg_memalloc*" parameter records the number of large memory allocations. The negative slope (-51.0073) of the correlation indicates that the amount of free user memory decreased along with increasing volume of large memory allocations. The *r-range* of this rule is [0.206937, 0.77661]. The correlation was not very strong (around 0.21) in system 2 while it was significantly strong in scenario 4 (around 0.78). Therefore, system performance degradation was accompanied by a significant increase in large memory allocation requests. However, we were not aware of strong correlations between *lg_memalloc* and other parameters. A likely explanation is that large memory allocations were caused by various subsystems, while each individual subsystem did not have a strong correlation with the parameter. We are going to employ regression tree analysis techniques [8] in the future to disclose such information. Nevertheless, at this point, we are aware of the existence of memory allocation bottlenecks in scenario 4.

Second, we consider the first rule in Table 5.3. The rule involves the *active checkpoint queue latch* and Oracle SQL*Net. According to [10] and [17], the Oracle DBMS links "dirty" memory blocks in the check point queues to improve the performance of checkpoint operation. A checkpoint can be highly resource intensive. More number of checkpoints reduces recovery time at the time of crash since less redo need to be reapplied, but causes negative impact on performance because of system overhead. In Oracle, various checkpoint queues are protected by the "*active checkpoint queue latch*" against concurrent accesses. The SQL*Net is a sub-component that ensures distributed client-client and client-server processing. The *r-range* of the rule indicates a significant leap of the correlation in system 4. We conclude that the checkpoint frequency of system 4 was too high and large amount of client requests from network had to block-wait for the completion of checkpoint operations. Obviously, we need to lower the system checkpoint frequency. This can be achieved by tuning the LOG_CHECKPOINT_INTERVAL parameter. Details about the parameter can be found in [13].

We note from the aforementioned experiment that the ADMiRe process is automatic, and the output is complete due to the application of data mining techniques. Nevertheless, domain knowledge is still very important to analyze the information provided by ADMiRe and make the right tuning decisions. As with most the data mining tools, the goal of ADMiRe is to discover useful knowledge, rather than to totally replace human experts.

6. CONCLUDING REMARKS

Automatic performance analysis tools are the only hope to cope with the ever-increasing complexity of computing systems. In this paper, we introduced a systematic technique called ADMiRe. Our approach consists of three steps. First, regression analysis is employed to summarize the sets of performance data, collected from various system states, into a more manageable size in the form of *regression rules*. These sets of rules are manipulated using rule algebra to reveal similarities, differences, and other information among different system states. Finally, the algebraic results are sorted and ranked to help the user focus on the more likely causes of the performance problems. The merit of this approach is threefold. First, data mining (i.e., regression analysis) is excellent for system performance analysis in that it is systematic, and the analysis results are complete. Second, ADMiRe can be used to examine performance of many software systems. Third, rule algebra is very flexible and powerful and can

potentially be extended to analyze output of various data mining techniques.

To assess the performance of ADMiRe, we used it to investigate the performance of an Oracle. Our experimental results indicate that useful information can be discovered by ADMiRe for effectively tuning the system.

Future work on ADMiRe involves the following directions. First, we are extending the system by employing more advanced data mining techniques to better analyze system performance. Second, more research needs to be done to adapt ADMiRe to generic data mining techniques. We are particularly interested in applying ADMiRe to supermarket transaction data sets to discover knowledge that points to better marketing strategy.

REFERENCES

[1] Allen J. F., "Maitaining Knowledge about Temporal Intervals", CACM26 (11) 1983, 832-843.

[2] Rakesh Agrawal, S. Gosh, T. Imielinski, B. Iyer, Arun Swami, "An Interval Classifier for Databases Mining Applications", Proceedings of the 1992 Very Large Databases Conference.

[3] Peter M. Chen and David A. Patterson, "A New Approach to I/O Performance Evaluation: Self-scaling I/O Benchmarks, Predicted I/O Performance", Pages 1-12 Proceedings of the 1993 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, May 10 - 14, 1993, Santa Clara, CA USA.

[4] Will Leland, Murad Taqqu, Walter Willinger, and Daniel Wilson, "On the Self-Similar Nature of Ethernet Traffic (Extended Version)", IEEE/ACM Transactions on Networking, Vol. 2, No. 1, pp. 1-15, February 1994. (From an earlier version of the paper in SIGCOMM 93, Sept. 1993.)

[5] Rakesh Agrawal, Ramakrishnan Srikant, "Fast Algorithms for Mining Association Rules", Proceedings of the 20th VLDB Conference.

[6] Maurice Houtsma, Arun Swami, "Set-Oriented Mining for Association Rules in Relational Databases", 11th International Conference on Data Engineering.

[7] Rakesh Agrawal and Ramakrishnan Srikant, "Mining Quantitative Association Rules in Large Relational Tables", Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data.

[8] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, Charles J. Stone, "Classification And Regression Trees", Chapman & Hall, 1998.

[9] Steven D. Gribble, Gurmeet Singh Manku, Drew Roselli, and Eric A. Brewer, Timothy J. Gibson and Ethan L. Miller, "Self-Similarity in File Systems", SIGMETRICS '98 Madison, WI, USA.

[10] Ashok Joshi, William Bridge, Juan Loaiza, Tirthankar Lahiri, "Checkpointing in Oracle", Proceedings of the 1998 VLDB. New York, 1998.

[11] Walter Willinger, Vern Paxson, and Murad Taqqu, "Self-similarity and Heavy Tails: Structural Modeling of Network Traffic. In A Practical Guide to Heavy Tails: Statistical Techniques and Applications", Adler, R., Feldman, R., and Taqqu, M.S., editors, Birkhauser, 1998.

[12] C. H. Cheng, A. W. Fu, and Y. Zhang. "Entropy-based Subspace Clustering for Mining Numerical Data", In SIGKDD, pages 84-93, 1999.

[13] Oracle, "Oracle 8i Tuning, Release 8.1.5", Part No. A67775-01, Oracle Corporation.

[14] Oracle, "Oracle Enterprise Manager Database Tuning with the Oracle Tuning Pack", Release 9.0.1, Part Number A86647-01, Oracle Corporation.

[15] Ken Gottry, "Successful Solaris Tuning", SysAdmin, the Journal for UNIX Systems and Administrators, 2001.

[16] Trevor Hastie, Robert Tibshirani, and Jerome Friedman, "The Elements of Statistical Learning, Data Mining, Inference, and Prediction". Springer 2001.

[17] Tirthankar Lahiri, Amit Ganesh, Ron Weiss, and Ashok Joshi, "Fast-Start: Quick Fault Recovery in Oracle", Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, 2001.

[18] Transaction Processing Performance Council, "TPC Benchmark C Standard Specification Revision 5.0", http://www.tpc.org/tpcc/spec/tpcc_current.pdf.

[19] Haixun Wang, Wei Wang, Jiong Yang, Philip S. Yu, "Clustering by Pattern Similarity in Large Data Sets", Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data.