RC 20083 (5/30/95)
Computer Science/Mathematics

IBM Research Report

Automating Performance Mangement Using Case-Based Reasoning

Joseph L. Hellerstein

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

Research Report

Automating Performance Management Using Case-Based Reasoning

Joseph L. Hellerstein
IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York 10598

May 15, 1995

IBM Research Division
Yorktown Heights, New York • San Jose, California • Zurich, Switzerland

Automating Performance Management
Using Case-Based Reasoning

Joseph L. Hellerstein
IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York 10598

May 15, 1995

ABSTRACT: Resolving performance problems in information systems is a time-consuming task for which automation can be of great benefit. Traditionally, this automation has been provided by rule-based systems. Herein is described an alternative approach that employs case-based reasoning (CBR). CBR matches new problem instances against previously solved problems (or cases), retrieving only those cases that are most similar to the problem instance. Case-based reasoning has the disadvantage of being less flexible than rules due to several hard-to-change decisions that are made at design time: the choice of attributes to match, the encodings used for attribute values, and the similarity metric employed. Also, compared with rules, CBR is less adept at explaining its conclusions. However, CBR is much more modular than rule-based approaches in that adding (or removing) a new case does not affect any existing case; in contrast, modifying a rule-base is, in general, a complex task. The modularity of CBR systems enables performance management automation that is custom tailored by end-users based on their own experience. Such automation can empower end-users to solve their own performance, thereby freeing scarce support staff. However, realizing these benefits in practice requires considerations for user-interfaces, software architecture, and business processes.

1. Introduction

The advent of low-cost microprocessors has spawned the wide-spread use of personal workstations to support applications such as spreadsheets, word processors, and presentation graphics. Unfortunately, the large number and variety of workstations and applications that must be managed poses great challenges for performance management. Addressing these challenges requires automation. To date, automation for performance management has been provided primarily by rule-based systems. Herein, we describe an alternative approach that employs case-based reasoning **(CBR)**, an artificial intelligence technique that matches new problem instances with previously solved problems.

What is required to make performance management scalable in today's heterogeneous, distributed computing environment? Our belief is that a first line of defense is the end-user. Specifically, it is highly desirable to have automation that allows end-users to easily solve performance problems that are under their control (e.g., the application mix, the tasks end-users perform). Doing so can shorten the time for problem resolution and free skilled analysts. When end-users cannot solve the problem, the second line of defense is the help desk. Here too it is essential to have automation, especially decision support systems.

To date, tools that automate aspects of performance management have employed rule-based programming, a technology that interprets *if-then* statements to encode problem solving strategies used by performance experts. Examples of rule-based systems for performance management include: [3], which makes recommendations on DASD tuning; [2], which analyzes MVS, CICS, and DB2 performance; [4], which provides recommendations for MVS performance problems; and [5], which advises on tuning VM computer systems. While some systems use case-based reasoning for diagnosing faults in information systems (e.g., [6] and [7]), we are unaware of others who have applied CBR to performance management.

Despite their widespread use, rule-based systems have significant deficiencies. First, it is difficult to develop rules that are applicable to diverse applications, workloads, and hardware platforms. As a result, the rule base tends to be complex. This complexity creates a significant maintenance problem since in order to add or change rules the interactions between them must be considered carefully. As a result, it is difficult for unsophisticated users of computers to add rules that customize performance management automation to their environment.

CBR has characteristics that make it an appealing alternative (or complement) to rule-based approaches. In essence, CBR uses a database of previously solved

Attribute	*Value*
Date	5/2/95
Time	10:15 am
CPU Execution	10%
CPU Wait	60%
IO Wait	20%
Memory	10%
Largest CPU Process	spreadsheet
Largest IO Process	backup
Largest Memory Process	backup

Figure 1: Data Used in Running Example

problems, or **cases**, against which new problem instances are matched. When a new problem instance is presented, the CBR system locates similar cases, ranking them by how closely they match the problem instance. Since each case is treated separately, new cases can be added without concern about interactions with existing cases. This feature allows end-users and relatively unsophisticated help desk personnel to add new cases.

The remainder of this paper is organized as follows. Section 2 describes case-based reasoning in more detail. Section 3 addresses user interface, software architecture, and business process considerations for employing CBR in practice. Section 4 compares CBR and rule-based systems. Our conclusions are contained in Section 5.

2. What is CBR?

This section describes the basics of CBR technology and illustrates how it can be applied to performance management.

To aid in this presentation, a running example is used. The context is a single user workstation from which process-level information can be extracted (e.g., as in [1] and [9]). We assume that the end-user is dissatisfied with the performance of a specific application, and (for simplicity) we assume that the application is single threaded (or that only one thread is of interest).

Fig. 1 contains the data collected for the performance problem considered.

- The date and time at which the problem occurred.

Attribute	*Value*
Business Cycle (CYCLE)	WEEKDAY
Work Shift (SHIFT)	prime
Bottleneck Resource (BRES)	CPU
Bottleneck Process (BPROC)	spreadsheet

Figure 2: Problem Instance (Transformed from Fig. 1)

- Measurements of the application under consideration, including its time spent in CPU execution, CPU wait, input/output (IO), and memory wait.
- The processes that consume the most CPU, IO, and memory over the monitoring period.

CBR matches new problem instances against previously solved problems. Problem instances are specified in terms of their attributes and values. Indeed, a problem instance can be viewed as a record in a relational database.

What attributes and values should be used for a problem instance? One could use measurement variables and their values as the attribute names and values. However, it is often desirable to transform the measurement data so that case retrieval is simplified. Fig. 2 displays such a transformation for the data in Fig. 1. The attributes of the problem instance are:

- Business cycle (**CYCLE**): a categorization of the date attribute by business activity, such as end-of-week (EOW), end-of-month (EOM), end-of-year (EOY), WEEKDAY, and WEEKEND processing.
- Work shift (**SHIFT**): a categorization of the time value into periods of the day, such as prime and off-prime.
- Bottleneck resource (**BRES**): The resource for which the problem application suffered the greatest delay during the measurement period.
- Bottleneck process (**BPROC**): The process that consumed the most of the bottleneck resource during the measurement period.

Note that the data in Fig. 2 can be constructed automatically from the data in Fig. 1 once some modest information about the business is available. Also, observe that

Case No.	*CYCLE*	*SHIFT*	*BRES*	*BPROC*	*Resolution*
1	*	*	Memory	*	Stop non-critical applications
2	EOW	off-prime	IO	backup	Stop backup
3	*	prime	IO	admin	Cancel software maintenance
4	*	prime	CPU	spreadsheet	Fix spreadsheet loop
5	EOY	*	CPU	auditor	Defer EOY processing

- BRES - Bottleneck resource
- BPROC - Bottleneck process
- * - Wildcard
- EOW - end-of-week
- EOY - end-of-year

Figure 3: Cases Used in Running Example

the transformations done exploit key elements of performance analysis: identifying the bottleneck resource and determining which process consumes the most of that resource.

A case is a problem instance plus its resolution. In particular, a case can be represented as a database record that consists of the attributes of the problem instance plus two additional attributes: the case number and the problem resolution. The collection of cases used by a case-based reasoning system is called the **case repository**. Fig. 3 displays several cases used in the running example.

With this background, the activities encompassed by CBR can be itemized:

- **case acquisition**: acquiring new cases and placing them into the case repository
- **case retrieval**: ranking cases based on their degree of similarity with a problem instance and retrieving the most similar cases
- **case generalization**: extending the case repository by creating new cases that are generalizations of cases in the repository

- **case adaptation**: transforming a case so that it better matches a problem instance

Herein, the focus is on case acquisition and retrieval. (Details on case generalization and adaptation can be found in [8].)

Case acquisition takes place when a problem instance is encountered for which there is no suitable case in the repository. Consider how case 1 in Fig. 3 is acquired when the case repository is initially empty. Suppose that the problem instance for case 1 is:

- CYCLE: WEEKDAY
- SHIFT: prime
- BRES: memory
- BPROC: word processor

Since no case is present, the problem is referred to an expert. Suppose that the expert determines that the reason for large memory waits is the presence of too many processes in the multiprogramming set. Further, suppose that the expert, in consultation with the end-user, resolves the problem by stopping several applications that are not critical to the end-user's task. Having solved the problem, a judgement is made about which attributes in the problem instance are relevant to the diagnosis found and the resolution reached. Here, it is assumed that only the bottleneck resource (*BPROC*) is of importance. So, values of the other attributes are set to a wildcard character, which we denote by '*'. Then a case number is assigned, and the problem resolution is recorded. This results in the case:

- Case no.: 1
- CYCLE: *
- SHIFT: *
- BRES: Memory
- BPROC: *
- Resolution: Stop non-critical applications

Case No.	*CYCLE*	*SHIFT*	*BRES*	*BPROC*	*Case Similarity Score*
1	1	1	0	1	3
2	0	0	0	0	0
3	1	1	0	0	2
4	1	1	1	1	4
5	0	1	1	0	2

- BRES - Bottleneck resource
- BPROC - Bottleneck process

Figure 4: Similarity Score Calculations for Running Example

Now consider case retrieval. Cases are selected based on their degree of similarity with the problem instance. This is quantified by a **similarity score** that is computed by comparing each case with the problem instance. Cases are sorted by descending value of their similarity score and then presented to the end-user.

How is the similarity score computed for a case? Typically, this is done by comparing the value of each attribute of the case with the value of the corresponding attribute of the problem instance. Then, the results of these comparisons are combined. Below is the specific algorithm used in the running example.

1. The similarity score for an attribute is set to 1 if the attribute's value for the case is either: (a) a wildcard (i.e., '*') or (b) equal to the attribute value of the problem instance. Otherwise, the score for the attribute is set to 0.

2. The similarity score for the case is the sum of its attribute scores.

Fig. 4 displays the results of applying this algorithm to the problem instance in Fig. 2 and the cases in Fig. 3. Consider case 1. It has the value '*' for the attributes CYCLE, SHIFT, and BPROC; so the score for each attribute is 1. The score for BRES is 0 since its value (i.e., Memory) differs from that of the problem instance (i.e., CPU). Summing the attribute scores, the similarity score for the case is 3. Note that case 4 has the largest similarity score and hence best matches the problem instance. Thus, the CBR system would rank highest the problem resolution "Fix spreadsheet loop."

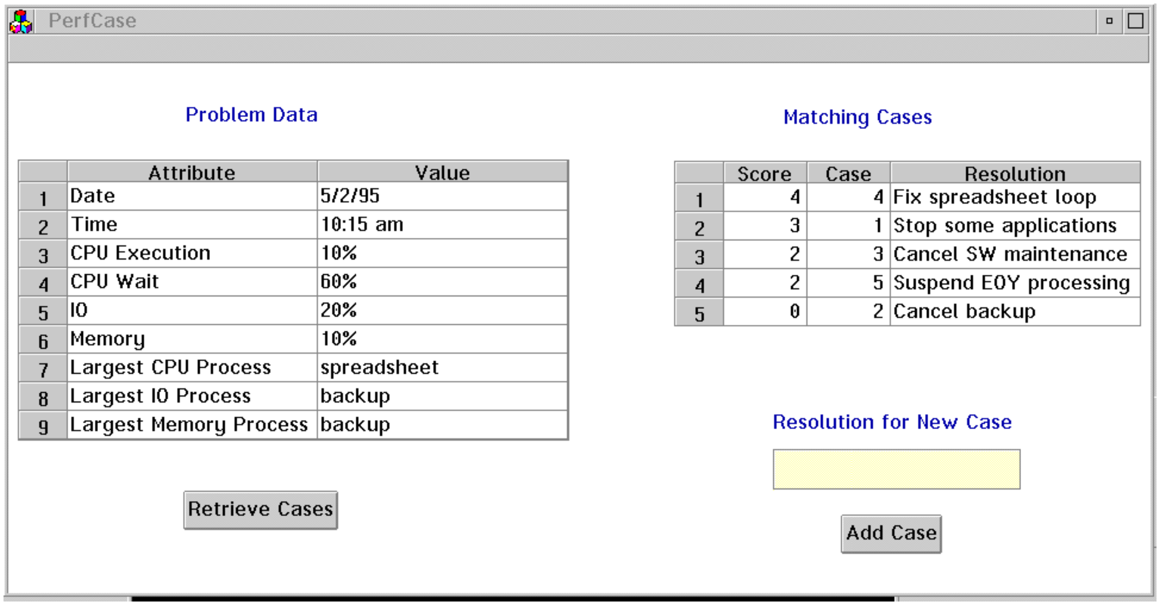

Figure 5: Sample User Interface for CBR-Based System

3. CBR In Practice

This section considers several issues related to employing CBR in practice. In particular, this section addresses considerations for: user interfaces, software architectures, and business processes.

The user interface requirements for CBR are modest. Problem instances must be entered; matched cases must be displayed; and there must be a way to add new cases to the case repository. Fig. 5 depicts a user interface that accomplishes these objectives. The leftmost table with the columns labelled *Attribute* and *Value* is used to specify problem instances. After data entry into this table has been completed, the *Retrieve Case* button is pressed. This causes the entries in the second table to be populated by cases that are sorted in descending order by their similarity scores. Adding a case is accomplished by specifying its attribute values in the first table, typing the case resolution in the *Resolution for New Case* field (at the bottom right

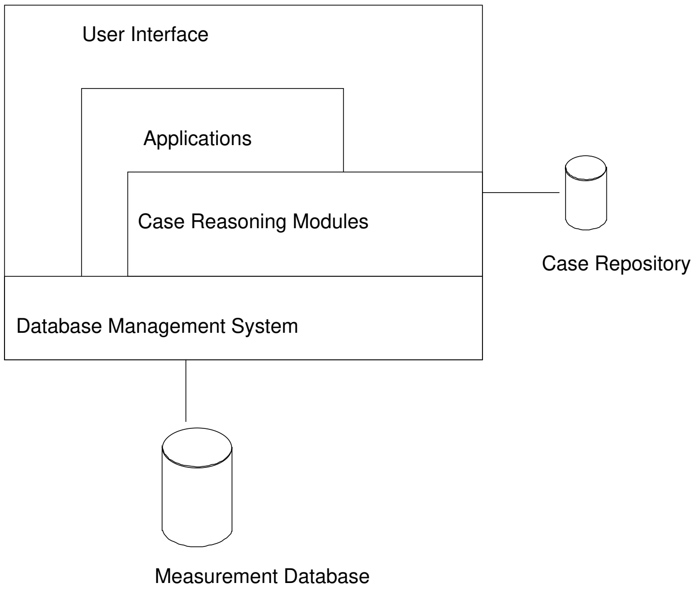

Figure 6: Architecture for a CBR-Based System

of the screen), and then pressing the *Add Case* button.

Addressed next is the architecture of a performance management application that employs CBR. So far, it has been assumed that the CBR system is an end-user application. More generally, there may be performance management applications that embed case-based reasoning (e.g., for capacity planning). An architecture that enables this is depicted in Fig. 6. Note that the CBR modules have access to the measurement database so that end-users need not enter data manually (e.g., they could instead specify a time interval during which a performance problem was present).

Last, we consider how a CBR tool can be incorporated into a business process for performance management. Fig. 7 depicts the flows in such a process. End-users enter problem data, either manually or specifying a query on the measurement database. Next, the CBR system converts this data into a problem instance (e.g., based on the kinds of transformations done on the data in Fig. 1 to produce Fig. 2). The user then

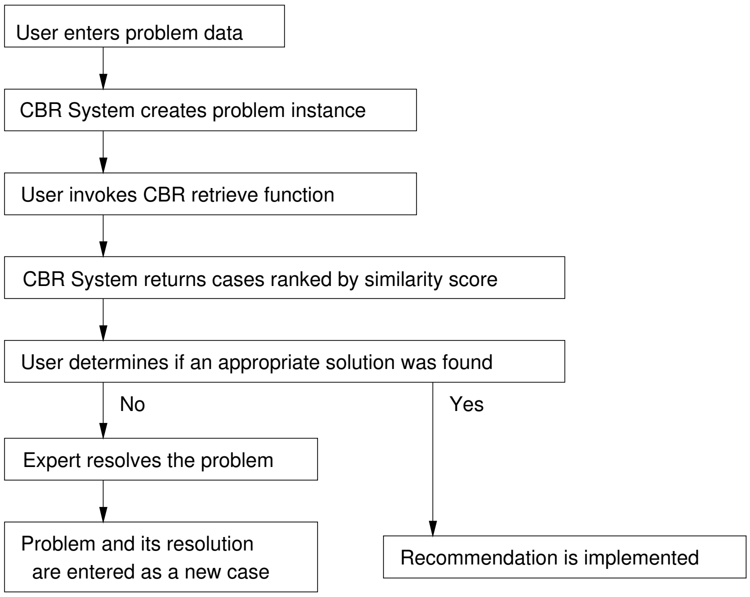

Figure 7: Steps In a Process for CBR-Based Performance Management

invokes the CBR system to retrieve cases that match the problem instance. The user examines the results reported to determine if there is an appropriate resolution to the performance problem. If there is, the recommended action is taken. If not, the problem is diagnosed manually and the resolution is recorded in the case repository.

4. CBR vs. Rules

This section compares CBR with rules for automating performance management.

We begin by returning to the deficiencies of rule-based programming that are cited in the introduction: (a) complexity resulting from interactions between rules; (b) difficulties with maintenance resulting from this complexity; and (c) difficulties with end-users adding rules as a consequence of (a) and (b). In essence, these issues arise because it is difficult to maintain a modular rule base without involving skilled analysts.

In contrast, CBR is inherently modular. Specifically, adding a new case does not affect the similarity score of any other case when a problem instance is matched against the case repository. Thus, it is reasonable to allow unsophisticated users of computers to augment the case-base. The only caveat is that appropriate change management controls are necessary to ensure quality and correctness if the augmented case-base is shared by others.

There are, however, a couple of aspects of today's CBR systems that make them less appealing than rule-based systems. First, the effectiveness of CBR systems is critically dependent on three somewhat difficult-to-change design decisions:

1. the choice of attributes for problem instances
2. the manner in which attribute values are encoded
3. how similarity scores are computed

To elaborate on (1), the data in Fig. 1 contain separate attributes for CPU Wait, IO Wait, and Memory Wait. However, the problem instance in Fig. 2 reduces these three attributes to the single attribute BRES, the bottleneck resource (i.e., the resource with the largest wait time in Fig. 1). Such a design works well if problems are readily characterized by a single bottleneck. However, if it is important to know the relative delay of two resources, a more elaborate design would be required.

The encoding of attribute values (item (2)) is also key to the effectiveness of CBR systems. For instance, date information in Fig. 1 is encoded into the CYCLE attribute by specifying business cycle information: end-of-week, end-of-month, and end-of-year processing. Doing so greatly improves the ability of the CBR system to find cases that took place during business activities similar to those for the problem instance. However, it may be that some performance problems have date dependencies that do not relate to business cycles (e.g., when the last software distribution took place). To accommodate such situations, a more elaborate encoding of attribute values is required and/or a more complex attribute scoring algorithm is needed.

The manner in which similarity scores are computed (item (3)) is also important. Section 2 describes an aggregated binary-scoring algorithm that is widely used in practice. Some changes to the similarity scoring algorithm are relatively easy to make. However, other changes are much more difficult to accommodate, especially those that require modifications to the attributes used or the manner in which attribute values are encoded.

Since decisions about items (1)-(3) must be made at design time, mistakes are difficult to remedy. In contrast, rule-based approaches are much more flexible in that it is relatively easy to add a rule that references an attribute that was not considered when the system was designed (assuming that data values for the attribute are available).

Another issue with today's CBR systems is that it is difficult to explain the logic behind the conclusions reached. This is a consequence of how cases are selected: by computing a similarity score. Thus, the only automated explanation possible is to explain how the score was computed. Such an approach conveys no insight into how a human would solve the problem, which may give the end-user less confidence in the recommendation produced by the CBR system. In contrast, rule-based approaches often provide useful explanations by listing textual descriptions of the rules invoked to produce the recommendation given.

Beyond the foregoing, additional issues arise when a CBR system allows multiple, unsophisticated users of computers to update the case repository. First, care must be taken to ensure consistency in the choice of attribute values. For instance, the attribute CYCLE in the running example has values WEEKDAY, EOW, EOM, and EOY. If a problem instance is entered with the acronym WDP instead of WEEKDAY, CBR matching is less effective since cases with a CYCLE value of WEEKDAY will not score as high as they should. Fortunately, this issue can be addressed by checking field values when problem instances are entered and when new cases are added.

A second consideration relates to maintaining the case repository. Traditionally, cases have been viewed as permanent truths that never become invalid. However, information systems, especially large ones, change frequently. Unlike rules, the modularity of CBR makes it trivial to update a case (assuming that attributes do not change) or to remove it. However, a much more challenging problem is determining *when* a case must be modified or removed.

5. Conclusions

The scale and diversity of today's distributed computing environment make it essential to provide automation for solving performance problems. To date, this automation has employed rule-based techniques. Herein is considered an alternative approach that makes use of case-based reasoning (CBR).

CBR matches new problem instances against a repository of previously solved problems or **cases**. Cases that score highest in this matching are reported to the end-user. Typically, similarity scores are computed by counting the number of attributes in the problem instance that are identical to those in the case.

Employing CBR in practice requires addressing considerations for user interfaces, software architectures, and business processes. As shown in Section 3, CBR's requirements are modest.

Compared with rules, CBR has the disadvantage of being less flexible in that several hard-to-change design decisions are critical to the effectives of the CBR system. These decisions include: the choice of attributes to match, the encodings used for attribute values, and the similarity metric employed. Further, compared with rules, CBR is less adept at explaining its conclusions.

CBR's appeal is simplicity and modularity. In particular, adding (or removing) a new case does not affect any existing case. In contrast, modifying a rule-base is, in general, a complex task. The modularity of CBR enables performance management automation that is custom tailored by end-users based on their own experience. Such automation can empower end-users to solve their own performance, thereby freeing scarce support staff.

References

[1] **Russ Blake** "Optimizing Windows NT," Microsoft Press, 1995.

[2] **Bernard Domanski**: "A PROLOG-based Expert System for Tuning MVS/XA," *Proceedings of the Computer Measurement Group*, 160-166, 1987.

[3] **Boole & Babbage**: *DASD ADVISOR Technical Backgrounder*, Boole & Babbage, Inc., 10BAB34.BGR, December, 1987.

[4] **Computer Associates**: "CA-MINDOVER," Computer Associates, 711 Stewart Avenue, Garden City, New York, 1993.

[5] **Siegmund Gralla** "An Expert System for VM Performance Analysis," *Research Report*, RC 12784, IBM Corporation, 1987.

[6] **Nialini Joshi**: "Knowledge-Based Network Diagnostics," *Proceedings of the Computer Measurement Group*, 8-17, 1992.

[7] **Lundy Lewis** "Case-Based Reasoning Approach to the Management of Faults in Communications Networks," *IEEE INFOCOM*, Vol. 3, 1993.

[8] **Janet Kolodner** "Case-Based Reasoning," Morgan Kaufmann Publishers, Inc., 1993.

[9] **X/Open Company Ltd.** "Systems Management: UMA Data Pool Definitions," Company Review Draft, (May 30-June 24, 1994), 1994.