Theoretical and Applied Informatics
ISSN 1896–5334
Vol.23 (2011), no. 1
pp. 37–53
DOI: 10.2478/v10179-011-0003-y

Adaptive improvement of resource utilization in goal-oriented multi-user systems

MACIEJ MŁYŃSKI

Association for Computing Machinery
maciej.mlynski@acm.org

Received 12 December 2010, Revised 1 February 2011, Accepted 9 March 2011

Abstract: The material presents a real problem inherent in the management of computer systems, namely that of finding the appropriate system settings and thus being able to achieve the expected performance. The material also presents a prototype which aims to adapt the system in such a way as to achieve the objective, defined as the application efficiency. The prototype uses a resource-oriented mechanism that is built into the OS Workload Manager and is focused on a proposed goal-oriented subsystem based on fuzzy logic, managing resources to make the best use of them, and pursuing translation to the use of system resources, including nondeterministic technology-related factors such as duration of allocation and release of the resources, sharing the resources with the uncapped mode, and the errors of performance measurement.

Keywords: adaptive performance improvement, heuristic algorithms, goal-oriented workload management

1. Introduction

Increasing volume and complexity of information systems currently in use requires continuous improvement of the method of management of IT resources. This forces scientists and architects to continue to work on new techniques that are more relevant to current needs.

Increasingly popular technologies such as virtualization, cloud computing, and dynamic resource allocation methods, allow the use of adaptive methods for managing resources. The ambiguity of measurements, overgrown possible parameters and the expectation of short response time to force scientists to the search for methods based on artificial intelligence, or on heuristic, bio-inspired and genetic algorithms.

The problem of efficient resource management is not entirely new. In the paper [1] the authors present the problems of decision-making, giving examples and stating that in

the 1970s these systems were based on decision-making spreadsheets, while during the 1980s optimization models were already being applied. In the 1990s decision-making systems were enhanced through the use of artificial intelligence.

In 2000s saw a significant increase in the complexity of computer systems and extensive use of virtualization mechanisms [14]. This resulted in work starting on the dynamic allocation of resources to improve the utilization of computing servers. The problem of resource management is closely linked to the computing environment, so work on this topic will have to be constantly updated. This work relates to managing the resources of the operating system z/OS (IBM Mainframe) [15], working in a distributed environment [5], in a Grid environment [6], and dealing with scheduling policies [7].

Presented in this article is a continuation of a previous paper [8] in which the issue was presented, a prototype solution for adaptive management of resources was proposed, and the goal-oriented approach was outlined. The prototype solution presented in the paper [8] was based on a knowledge base. Such a solution has its advantages and disadvantages. In some cases, for example, the program changes its characteristics when executed, and it is also quite expensive to build a knowledge base.

The paper [8] suggested using heuristic algorithms to eliminate the costly process of building a knowledge base. Application of heuristic algorithms in the management of resources is also necessary in cases where the characteristics of the course program are unknown. This was our motivation to continue the research, the current paper presents a problem and a working prototype solution based on heuristic algorithms. The algorithms which were presented here and the process of their description was inspired by the paper [2, 3, 4], in particular, fuzzy logic defined by Professor Zadeh [2].

The environment used in the experiments is exactly the same as in the paper [8]. The aim is not to add a new module to the operating system, but to use the built-in operating system's resource-oriented Workload Manager. On top of the subsystem a goal-oriented module has been added [9], in such a way that it dynamically adjusts resources in order to achieve better use of resources and achieve the objectives in advance of the expected requirements of the run-time applications.

Section 2 presents related work. This section refers to an article about the goal-oriented approach and heuristic algorithms. Similar work has addressed exactly the same problem, i.e. translations from goal-oriented to resource-oriented using heuristic algorithms could not be found. Section 3 describes the problem, while section 4 analyses it more deeply. In this section, comparisons between solutions based on a knowledge base and based on heuristic algorithms are presented. Each method has its own advantages and disadvantages; depending on requirements both might be used successfully. The methods using a knowledge base are perfect for solutions which have the same characteristics in all conditions, whereas a dynamic and non-deterministic environment needs more attention and, for this, algorithms which can deal with unclear information are

more welcome. Section 5 describes the evaluation of the heuristic algorithms proposed for this problem. Section 6 presents the results of the experiments which were made using the developed prototype. A summary can be found in section 7.

2. Related Work

Research on multiple scheduling on parallel and distributed computer systems has been in progress ever since the computer systems have been developed. The research is not yet finished, as it still needs to be improved and needs to use new methods according to the environment in which the computer system is to be used. In [7] the authors developed a new method for goal-oriented job scheduling policies. The motivation for their work was the fact that traditional job schedulers are configured with many parameters for defining job or queue priorities. Using many parameters seems flexible, but, in reality, tuning the values is highly challenging. To simplify resource management, the authors [7] proposed goal-oriented policies which allow system administrators to specify high-level performance goals rather than tuning low-level scheduling parameters. They used first-come-first-served (FCFS)-backfill algorithms [16] as their baseline. Many papers have proposed priority functions for improving FCFS-backfill, and some papers studied the performance impact of giving more than one reservation [7]. The methods that have been developed, such as large slow-down first (LXF)-backfill, which gives priority to the job with the largest expansion factor, significantly improve the average slowdown and average wait but give a worse maximum wait when compared with FCFS-backfill. The work most relevant to [7] is [17] which proposes adaptive policies. Under these adaptive policies, different backfill policies may be used during different periods of time.

In [5] the authors focus on multiple data center sites, with each handling workloads according to an enterprise-level strategy. The individual sites are federated and autonomous. Each workload has a Service-Level Agreement (SLA). Developed in [5], an analytical tool is capable of evaluating complex policies on a large-scale system and permits independent policies for each site so that policy makers can quickly evaluate several alternatives and their effects on the workloads before developing them.

In [7] two objective models are studied: Lexical and Eq-Tradeoff. Using the objective models, the authors [7] define a set of goal-oriented policies. In [5] the authors superimpose a queuing network model on the finite state model to annotate the state transitions with their probability distribution functions. In [6] the authors apply the Kalman filter and Savitzky-Golay smoothing techniques to train a sequence of confidence windows and present a new adaptive hybrid method (AHModel) for load prediction guided by trained confidence windows.

Several workload prediction methods measure mean-value or median performance, or use auto-regression (AR), polynomial fitting, the Markov model, or seasonal varia-

tion to predict performance with various look-ahead times [6]. The research projects presented in [5, 6, 7] are based mostly on theoretical foundations. Also, scientific workloads do not take into account the exact characteristics of applications used in real-live environments widely found in commercial institutions such as banks, insurance or telecom companies. Before introducing the suggested algorithms, some modifications to the operating systems are required.

The research presented in this article is based on what has already been implemented in operating systems' resource-oriented mechanisms which are working well. The problem is how to translate a goal-oriented approach into resource-oriented policies. In [10] an analysis was presented in which an application's performance could be influenced by the operating system's internal processes such as memory swapping, re-paging, or I/O threshing. In [11] the author show that incorrect setting of class assignment rules and policies can lead to low global system efficiency. In [12] the author demonstrates similar issues with a partition-load manager and a micro-partitioning environment.

Some similarities to this research can be found in [15], but the authors of [15] based their work on a different environment and pretended to change the implemented resource-oriented Workload Manager into a goal-oriented one internally in the operating system. This concept can cause the operating system to be dedicated only to a narrow area of applications with performance characteristics that are suitable to the Workload Manager implementation.

The concept of our work is to use artificial intelligence to perform dynamic goal-oriented workload operations. By "artificial intelligence" we understand that heuristic algorithms are proposed, some of them being well defined such as Fuzzy Logic [2], dynamic optimizations [1], or evolutionary computation [4]. In [3] the authors noted that most applications of evolutionary algorithms (EA) deal with static optimization problems. However, in recent years there has been a growing interest in time-varying (dynamic) problems, which are typically found in real-world scenarios.

In this article we propose to use existing built-in operating system workload mechanisms and to use their control modules, based on heuristic algorithms, to translate the goal-oriented approach into their policies.

3. The goal-oriented approach

The Workload Manager system is divided into N classes. Each class might contain its own programs. The programs can vary its characteristics in different classes and can be assigned to several classes at the same time. Individual classes can contain different programs. Each class can be adjusted according to its own set of parameters, such as hard maximum of memory consumption, hard maximum of processor utilization, and hard maximum of utilization of the I/O channels.

Classes can have more parameters, such as a hard, as well as soft, minimum of utilization of the resources. "Soft" means that the value may be exceeded if the other classes do not use their computing power. Due to the complexity of the problem we propose to control the use only of the hard maximum value for systems of I/O, CPU and memory. These simplify the nature of the tests, but the results are also accurate for complex systems.

A workload management tool's built-in operating system is usually resource-oriented [9]. In this approach each class of resources is focused on resource allocations dependent on the amount of resources already used by the class. In Workload Manager in AIX, low level functions allocate the resources depending on the class's utilization [9]. The utilization is calculated online. Each class has its own color. Red means that it is fully utilized and does not need resources, whereas white means that it does need more resources. During process creation, the operating system is deciding whether the new resources should be allocated or not. The online class coloring effectively improves the managed workload decision process. To avoid killing the processes, in case of the need to reduce resources, priorities are decreased for all processes classified according to the particular class.

This mechanism concerns the amount of resources allocated to particular classes. In dynamic systems, the characteristics of the application change depending on external conditions. The resource-oriented mechanism can cause problems in that even when the resources are allocated properly, the application's response time is not good enough. In that case the resources must depend on the response time. The problem is how to find the function for translation between response time (objective) and the proper amount of resources needed to match the objective.

Fig. 1 shows a diagrammatic view of the problem. On the left we have the system resources allocated to the Workload Manager classes. Each class can be created, deleted and reconfigured online. Processes that are allocated to particular classes are assigned by previously defined class assigned rules, which can also be reconfigured online. Some processes can be allocated to pre-defined system classes such as System, Shared or Unmanaged. The class contains processes that are responsible for memory re-paging, filesystem caching or scheduling. Resources allocated to this class can also have an impact on overall system efficiency. Some groups of resources are also shared between classes, for memory pages allocated to Virtual Memory Manager (VMM).

The efficiency on this diagram, termed *Eff*, is defined as application efficiency. Workload Manager cannot measure the efficiency of an application automatically. This is done by a user-defined procedure that is dependent on the application's characteristics.

Each of the applications, installed in their respective classes, has its own characteristics measured as *Eff N*, defined as performance. The performance for each application can be defined separately, but it is measurable on each occasion. This may be the num-

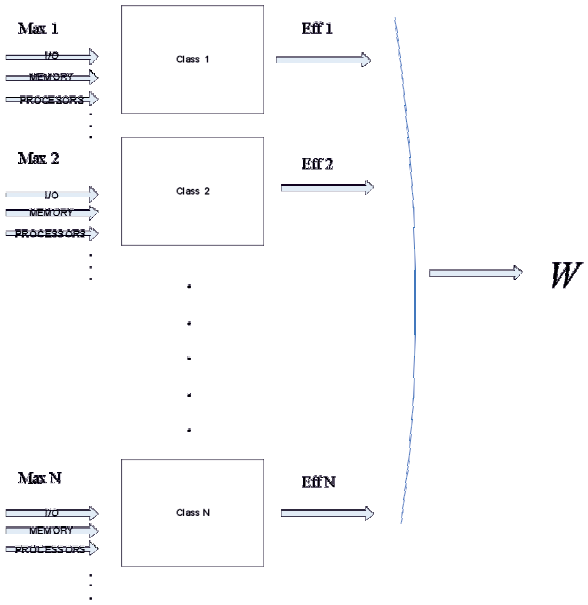

Fig. 1. Schema of the goal-oriented approach. Classes in Workload Manager in AIX operating systems are resource-oriented. The functions Eff(1,N) and W are goal-oriented

ber of transactions processed per unit of time, records in a database, or response time to database queries.

The entire automatic steering mechanism is controlled by the imprecisely defined function W which controls that server is used as best as possible. One of the assumptions is to use a server that is as efficient as possible. Requirements for *Eff 1 .... Eff N* may change over time, since they are generally understood to change according to user requirements.

The problem is how to choose the parameters *Max 1 .... Max N*, so that the user's requirements are met. Parameters *Max 1 ... Max N* define the capacity of the server, but performance depends on many parameters. Sometimes increasing the amount of resources does not improve application performance.

Finding the parameters must also be realized in a short time, but the value of the efficiency *Eff 1 .... Eff N* is not necessarily precise and may be adjusted later if necessary. It is possible to introduce a parameter to the system saying that the class is now desirable to use. For instance, assigning 20% of memory consumption does not have the immediate result that the application will use this amount of memory. Processes must be previously expropriated. In workload management solutions in operating systems, in order to reduce resources, the process priority is decreased or the scheduling priority

significantly changed by increasing penalty points or increasing the time during which the penalty for using a processor will be canceled.

In [11] the authors set up Workload Manager by using an ad-hoc method, while in [8] the knowledge base was created by using most of the combinations, because the parameters in Workload Manager can be of every permutation. Our approach was to eliminate any combination where utilization of the system can be limited by Workload Manager. Anything sought from the set of parameters can be excluded where the value:

$$\begin{aligned} &\sum_{i=1}^{N} Max_i I/O \neq 100\%, \\ &\sum_{i=1}^{N} Max_i MEM \neq 100\%, \\ &\sum_{i=1}^{N} Max_i CPU \neq 100\% \end{aligned} \tag{1}$$

The scenario under consideration was to store all these parameters in the knowledge base. By contrast, such a scenario is not suitable for real applications because it requires an examination of their characteristics in all permutations of settings. There is also a problem with this solution, in that the time needed just to generate all the permutations for a few classes could be extensive. It was therefore sensible to use heuristic algorithms.

4. A workload-prediction solution

The considered case of resolution was a B50 pSeries machine with one processor, PowerPC_604e, 375 MHz, 1024 MB RAM, and two internal SCSI drives on the SCSI controller Wide/Fast-20. The server configuration was selected because of needs to have a better possibilities to measure a program execution time. The advantage of this solution is that, with such a slow processor, the speedup of the test program is more apparent.

Actually, we do not know if the application performance is proper, or if the performance is impaired by stabilizing Workload Manager. The tested solution assumes that waiting about five seconds makes the measurement stable. A real environment should be explored in which the application performance is changing and where the next several measurements give the same results, only assuming that it is the correct application performance under the given working conditions and Workload Manager settings.

The average time necessary to set up these classes is the time required to build a knowledge base and the time to read the approximated values.

However, as well as attempting to find the settings for the CPU, it would also be necessary to find the settings for memory and I/O channel bandwidth. Equation 2 should be used to estimate the number of combinations for Workload Manager:

$$K_n = iCPU_n * iMEM_n * iWe/Wy_n * ... * iX_n \tag{2}$$

The parameters are defined as follows:

K_n	The total number of combinations, where n is the number of classes.
$iCPU_n$	Number of combinations to the limits of the processors, where n is the number of classes
$iMEM_n$	Number of combinations to the limits of memory, where n is the number of classes.
iWe/Wy_n	Number of combinations to limit the I/O channels, where n is the number of classes (We/Wy means I/O).
iX_n	Number of combinations of additional settings, where n is the number of classes.

Below are a few examples of how much iterations must be used in case of needs of using methods with knowledge base:

Example 1. Number of combinations for three classes (app, prod24, prod) in the areas of hard limits on the maximum utilization of processors, memory, and examples of I/O. K_n = 4851 * 4851 * 4851 = 114 154 707 051

Example 2. Number of combinations for two classes (app, prod) in the area of I/O and additional system settings. For instance, two-state characteristics of a file system such as CIO (Concurrent I/O) would be as follows: K_n = 97 * 97 * 2 = 18 818

Example 3. Number of combinations for three classes (app, prod24, prod) in the areas of hard limits on the maximum utilization of processors, memory, and examples of I/O, and the twelve-setting performance applications K_n = 4851 * 4851 * 4851 * 12 = 1,369,856,484,612

In Example 3, if we get a stable measurement lasting 20 seconds, then the time to build a knowledge base would take approximately: T ≈1,369,856,484,612 * 20 sec. = 27,397,129,692,240 sec. = 7,610,313,803.4 hours = 317,096,408,475 days = 864,022.9 years

To prove how the efficiency of the application is dependent on the limits, some tests must be undertaken. Fig. 2. and Fig. 3. show the relationship between setting limits and the Workload Manager application performance. Workload Manager settings are measured in terms of % CPU utilization. Application performance is measured as the duration of the short program – measured in seconds.

Fig. 2. and Fig. 3. show that application performance is not closely linked to the Workload Manager settings. The Workload Manager settings are marked on the graph as straight lines whereas the application efficiency is not linear.

In Fig. 2 the first *WLM_PROD_Setting* is set to 1. The *Efficiency_PROD* value changes anyway. The line is high, because the response time in this case is long. The class is low on resources (1% CPU), so the response time is the longest of all the rest of the program execution times, at approximately 40-60 seconds. In Fig. 3,

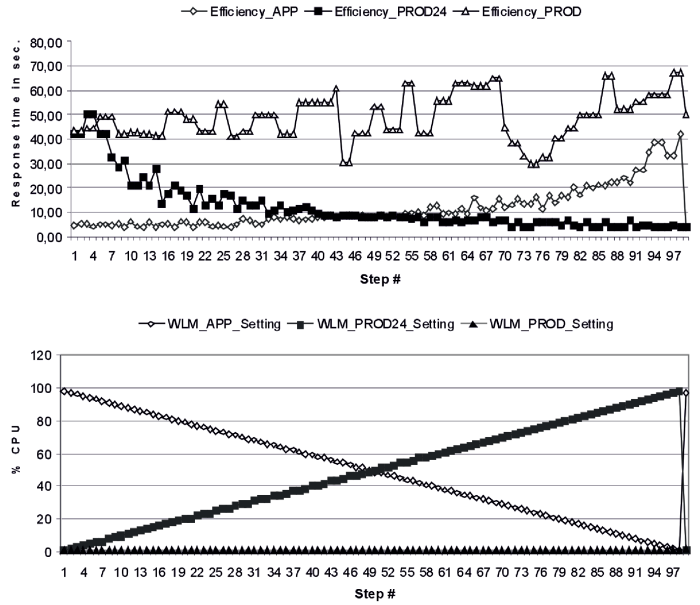

Fig. 2. Application efficiency – first one hundred samples

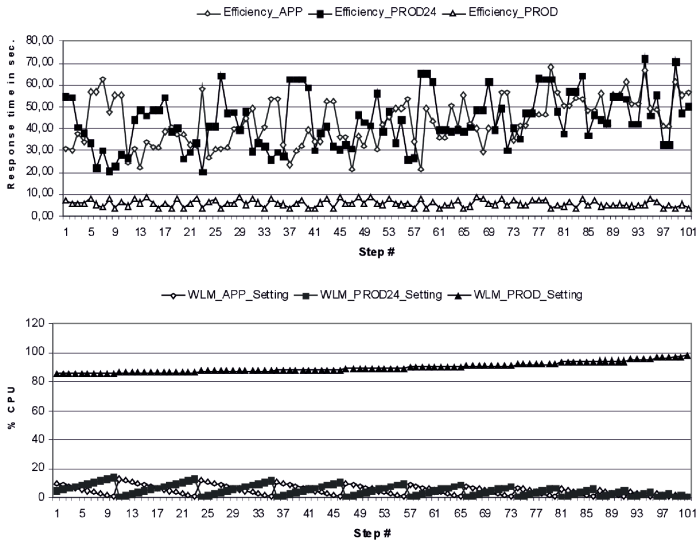

Fig. 3. Application efficiency – last one hundred samples

WLM_PROD_Settings change slowly from 90-100%, but the changes of settings of the two other classes are very frequent.

The programs and scenarios in the experiments in this paper were specially prepared to show the dependency and to enable any research to be possible, because for the research we needed to have a repeatable environment. In [10] the author presented a real problem based on a production banking environment based on Oracle databases. Highly-loaded and missing critical systems have some other characteristics which can have also influences here. In production databases most data are stored in Unmanaged and System classes.

5. Heuristic algorithm

While we were studying the problem as experienced and developing our proposed prototype we were not decided on which heuristic algorithm could be used. We first went through the requirements and the expected results. In [2] the authors defined a class of computation techniques named evolutionary computation (EC). The best known algorithms in this class include genetic algorithms, evolutionary programming, evolutionary strategies, and genetic programming [2]. Based on this statement and described in the framework of [2], we went through a process of definition of our own heuristic algorithm for the previously described problem. The most important components of any heuristic method are:

a) The creation of a data structure describing the feasible solution; a vector of numbers, or some other data structure.

b) The creation operators transform one solution into another.

c) The creation of an evaluation function, which is a measure of the quality of the solution.

With these ingredients ready, it would be easy to implement most of the heuristics. The definitions of these above components are as follows:

Ad. a) The data structure describing a feasible solution is a set of permutations of Workload Manager settings. In our case, for the three classes for example, it would be:

```
<setting values WLM>, ->  <application performance>
   90,  5,5,90,5,5,90,5,5,-> performance
   89,  6,5,90,5,5,90,5,5,-> performance
   88,  7,5,90,5,5,90,5,5,-> performance
   87,  8,5,90,5,5,90,5,5,-> performance
   86,  9,5,90,5,5,90,5,5,-> performance
   85,10,5,90,5,5,90,5,5,-> performance
   84,11,5,90,5,5,90,5,5,-> performance
```

They are, therefore, the total number of ranges 1 – 100 for classes of the Workload Manager, or another, but the known range of the set of additional system settings. Maximum and minimum application performance is also well known. Typing unrealistic values to the algorithm will not switch to starting the process of changing the system parameters. For the experimental environment we used rules that were as simple as possible. In production and real-time environments the rules must consider elements like the Unmanaged classes or time for relocating resources. In Workload Manager, inside the operating system, this can be acceptable, but in the case of using the developed prototype in the virtual environment described in [12], such as *micro-partitioning*, *PowerVM*, *PartitionLoad manager*, or *PartitionMobility*, the time for resource reallocation is much more critical.

In our lab environment, the developed operators for the heuristic algorithm are as follows:

```
   IF performance IS very poor THEN significantly increase
the resources, reducing other resources
   IF performance IS poor THEN Increase resources,
reducing other resources
   IF performance IS good THEN do not manipulate
```

The performance for one class may be appropriate, while the performance for other classes may be weak or very weak. Increasing resources may also in some (admittedly very rare) cases reduce performance, whence an additional rule should be added to restrict the increase in amount of the resources. This is a modification of the standard Fuzzy Logic rules defined by Professor Zadeh. The limit when a new resource should not be added is dependent on dynamic factors that depend on server architecture, application characteristics and some other unknown factors. As described in [12] the hardware architecture can be changed online. A correlation between the changes and the modified Fuzzy Rules must be defined. This is a topic for future research on this subject. In this article we have focused on a definition of the source of the problem. We are using the Fuzzy Rules in as simple a form as possible for that reason.

Ad. b). All vectors <*wlm settings*>, <*application efficiency*> can be found by successive trials. However, the action is often laborious and not feasible in real environments, such as when an amount cannot be credited to one's customer in
a banking system. The main objective of the heuristic algorithm is, therefore, to find assignments <*wlm settings*> that are unknown to the known <*application efficiency*>.

Ad. c) The evolutionary function is <*application efficiency*> which may approach to the given exact value of such a execution of indicated procedure or stored procedure in given time, or approach to some approximation such as the number of simultaneously working users while the built-in procedure is performed with a median time of

10 seconds. The number of simultaneously working users can depend also on external circumstances or might be given in certain parameters like vary from 1000 to 1100. In this paper, the evolutionary function was defined as the duration of the execution of the short program, measured in seconds.

6. Results of the experiments

Fig. 4 presents the prototype. Each of the Workload Manager classes has its own methods for measuring the local application efficiency described as *Eff 1*, *Eff 2*, and *Eff 3*. Function W controls the server utilization. Incorrect settings for local classes can result in the server not being fully utilized. Function W is for controlling the incorrect behavior. All measurements are collected by Analyzer, which is based on Fuzzy Rules and has capabilities for deciding whether and how the workload classes can be reconfigured. The reconfiguration is done by executor, which is able to set the correct resource-oriented rules for the Workload Manager subsystem.

Fig. 5 shows the results from use of the proposed method. On the horizontal axis is the time duration of the exercise experiments. On the vertical axis is shown the setting of Workload Manager classes as a percentage of CPU utilization. The lines are marked *ParamAPP*, *ParamPROD24*, and *ParamPROD*. The vertical axis shows the application performance - measured as the duration of the program.

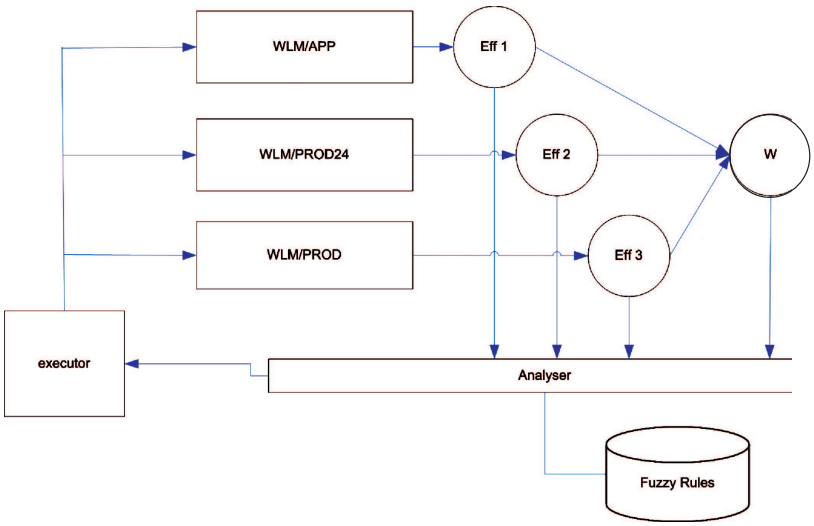

Fig. 4. Prototype based on Fuzzy Logic

Shorter duration of the program means his better efficiency. In Fig. 5 parameters *GAPP*, *GPROD24* and *GPROD*; described also as the expected execution time for the

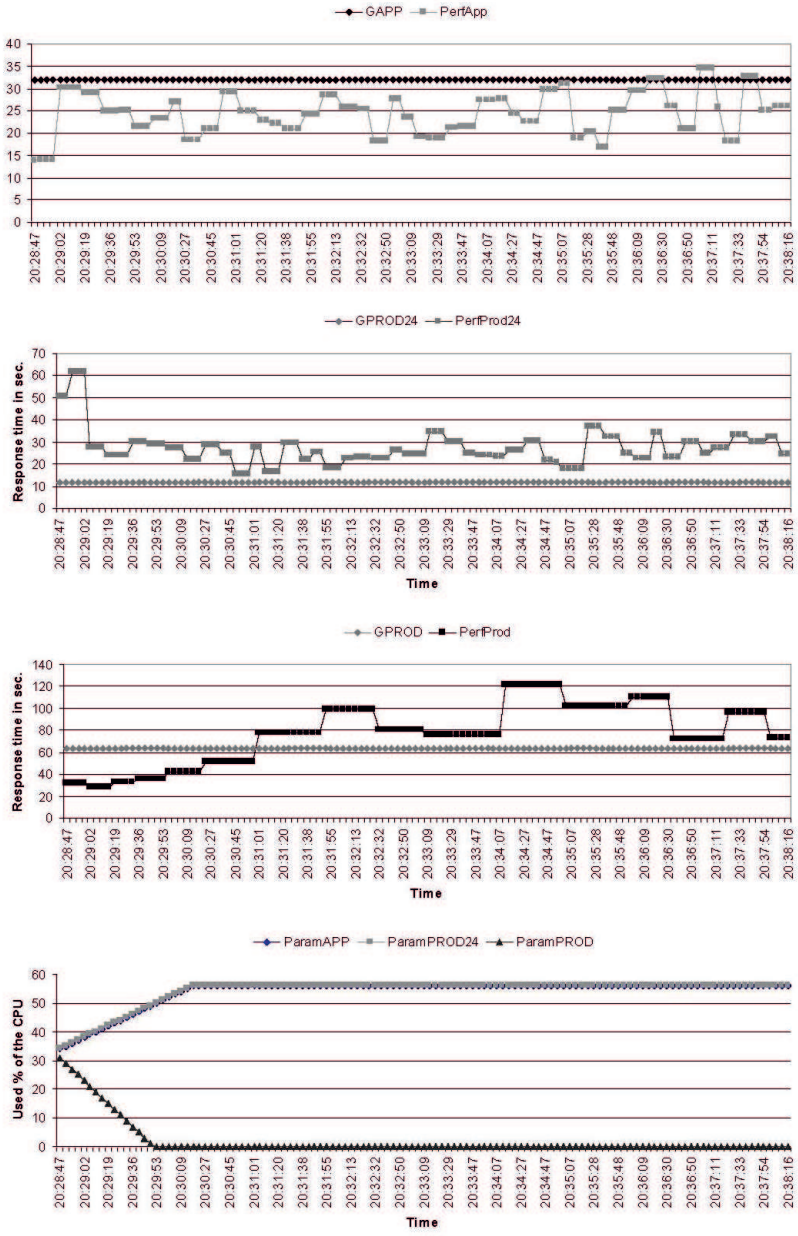

Fig. 5. Application efficiency

program is the aim of the efficiency. The actual performance of the examined program in Fig. 5 is described as *PerfAPP*, *PerfPROD24* and *PerfPROD*. Fig. 5. shows scaled application performance. For better visibility, *Perfx* and *Gx* values have been multiplied by two. In Fig. 5 *ParamAPP* and *ParamPROD24* have the same values.

As the chart on Fig. 5 shows, finding values close to those expected was fairly quick – much faster than building a knowledge base. Adding more complex decision rules should improve the accuracy and appropriateness of solutions, and help regulate the boundary condition. Despite the elements which can still be improved (a prototype was presented, and not a ready-made solution), the method using fuzzy logic gives very good results in the regulation of the parameters of Workload Manager.

Tab. 1 shows comparison between the two methods. The three left columns is about Methods with knowledge base, described in [8], and the three right columns about heuristic algorithm described in this paper.

Tab. 1 shows that the time it takes to find the performance is the time to generate the permutations, plus the time to measure the application performance, and the time needed to stabilize the environment. Stabilizing the environment is desirable so that, after the changes to the Workload Manager, it must then dispossess the resources so as to achieve the expected state. Stabilizing can be faster if the changes are close to each other, but slower when the changes are far away.

The number of permutations in Tab. 1 was reduced by combination where the expected occupancy of resources was other than 100%. We can also consider the possibility of eliminating the boundary conditions, assuming that the class of occupancy of less than 5%, for example, may be unstable.

	Method with knowledge base			**Heuristic algorithm**		
	Number of permutations	**The time needed to find the performance of applications in any combination**	**The average time required for setting classes**	**Number of permutations**	**The time needed to find the performance of applications in any combination**	**The average time needed to set the class**
1class	100	100*TS	Not measured	100	≈ 2 sec.	≈ 2 sec.
2classes	97 ##	97*TS	Not measured	97	≈ 2 sec.	≈ 2 sec.
3classes	4851	4851*TS	≈ 5 days $#	4851	≈ 2 sec.	≈ 2 sec.
4classes	156849	156849*TS	Immense $$	156849	Not measured	Not measured
TS = 20 sec. for stabilization + measurement time for Application response measurement ## – In theory 99 *$$ – In theory, about 36 days* $# – About 5 days to create a knowledge base + 0.07 seconds to read data						

Tab 1. Comparison of the two solutions. The data named "methods with knowledge base" are based on [8] while the "heuristic algorithm" data are described in this paper

7. Summary

Presented in this article is a prototype solution, based on fuzzy logic, to eliminate the process of building a knowledge base. The prototype for adaptive workload management based on the knowledge base was based on the prototype presented in the [8]. After creating the rules for decision making, finding the Workload Manager settings took two seconds. This is a good result, since the time needed for construction of the knowledge base for the three classes is approximately five days.

A continuing problem is the dynamic creation of decision rules in the case of adding or removing classes, or a substantial change in the characteristics of the application. The presented solution has not been tested for applications that have certain properties to adapt to the environment. Such behavior could impede the smooth finding of the correct settings for Workload Manager.

Problems with workload management at the operating system level are complex. On the one hand, the algorithms should be accurate for the hardware architecture and application, on the other hand the algorithm must be quick. Any latency, even of milliseconds, can causes huge performance degradation. Introducing a heuristic algorithm for this purpose seems to be the natural way to resolve the problem of balancing accuracy and efficiency.

Sections 4 and 5 of this article describe the need for modification of the Fuzzy Logic rules defined in [2]. It is not enough to create rules and use them over a long period of time. The rules must be automatically modified, depending on environmental conditions and specific requirements for the expected efficiency.

In [13] the authors described Service Oriented Architecture, which is still new and not clearly defined. The architecture is suitable for targeting operations which are taking place in ICT departments in commercial companies right now. The architecture is dependent on services which must be provided with defined quality. An element of that quality is the response time. How to measure the response time is still not clearly defined, but the definition is required in order to have proper workload management at the operating system and hardware level. We plan to explore the measurement techniques including commonly used for measure customer satisfactions and defects from business point of view like Six Sigma (6δ).

Current techniques of resource allocation and dynamic changes in system architecture offer new opportunities for the development process. In the paper [12] examples were given where even small changes in hardware architecture have a significant impact on application performance. The application of heuristic methods for addressing the virtual environment is another field in this area in which work will continue.

References

1. Z. Michalewicz, M. Schmidt, M. Michalewicz, C. Chiriac: *Case Study: An Intelligent Decision-Support System*, IEEE Inteligent Systems, Vol. 20, No. 4, 2005.
2. L. A. Zadeh: *Fuzzy Logic*, Neutral Networks, and Soft Computing, Communications of the ACM, Vol. 37, No. 3, 1994.
3. R. K. Ursem, T. Krink, M. T. Jensen, Z. Michalewicz: *Analysis and Modeling of Control Tasks in Dynamic Systems*, IEEE Transactions on Evolutionary Computation, Vol. 6, No. 4, pp. 378-389, 2002.
4. Z. Michalewicz: *Heuristic Methods for Evolutionary Computation Techniques*, Journal of Heuristics, Vol.1, No.2, pp. 177-206, 1995.
5. M. Srivatsa, N. Rajamani, M. Devarakonda: *A Policy Evaluation Tool for Multisite Resource Management*, IEEE Transactions on Parallel and Distributed Systems, Vol. 10, No. 10, 2008.
6. Y. Wu, K. Hwang, Y. Yuan, W. Zheng: *Adaptive Workload Prediction of Grid Performance in Confidence Windows*, IEEE Transaction on Parallel and Distributed Systems, On-Line, 2009.
7. S. Chiang, S. Vasupongayya: *Design and Potential Performance of Goal-Oriented Job Scheduling Policies for Parallel Computer Workloads*, IEEE Transactions on Parallel and Distributed Systems, Vol. 19, No. 12, 2008.
8. M. Mlynski: *Automatic Adjustment of the Settings of Workload Manager for Adaptive Performance Management*, Theoretical and Applied Informatics, Vol. 21, No. 1, pp. 37-56, 2009.
9. M. Mlynski: *Dynamic Resources Allocation in AIX 5L*, XII Conference on Real Time Systems, WKL, pp. 247-256, 2005.
10. M. Mlynski: *Analysis of Using an AIX Dynamic Resource Allocation Mechanism to Describe a Utility Level of Server in Oracle Data Bases Environment*, ZN Pol. Sl. s. Informatica, Vol. 26, No. 3 (64), Gliwice, 2005.
11. M. Mlynski, P. Rumik: *Examination of Usefulness of Workload Manager in AIX 5L to Improve Efficiency of Application in MySQL Data Base*, II Conference on Databases, Applications and Systems. WKL, 2006.
12. M. Mlynski: *The Influence of the IBM pSeries Server's Virtualization Mechanism on Dynamic Resource Allocation in AIX 5L*, Scalable Computing: Practice and Experience, Vol. 10, No. 2, pp. 189-199, 2009.
13. N. Alur, M. Mlynski, S. Balakrishnan, O. Shure, Z. B. Cong: *SOA Solution Using IBM Information Server*, IBM International Technical Support Organization, San Jose (USA), IBM Press, 2007.
14. C. Matthys, M. Mlynski, N. Tollet, G. Barbati, H. Chauhan, B. Dierberger, R. Marchini, H. Wittmann: *Planning, Installing and Using the IBM Virtualization Version 2.1*, IBM International Technical Support Organization, Poughkeepsie (USA), 2006.

15. J. Aman, C. K. Eilert, D. Emmes, P. Yocon, D. Dillenberger: *Adaptive Algorithms for Managing a Distributed Data Processing Workload*, IBM Systems Journal, Vol. 36, No. 2, p. 242, 1997.
16. D. Lifka: *The ANL/IBM SP Scheduling System*, Proc. First Job Scheduling Strategies for Parallel Processing (JSSPP '95), Apr. 1995.
17. B. Lawson and E. Smirni: *Self-Adaptive Scheduler Parameterization via Online Simulation*, Proc. 19th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '05), Apr. 2005.

Adaptacyjne polepszanie wykorzystania zasobów informatycznych w wieloużytkownikowych systemach komputerowych

Streszczenie

Artykuł przedstawia realny problem występującego w zarządzaniu systemami informatycznymi polegający na dynamicznym znajdowaniu odpowiednich ustawień systemu, dzięki którym można osiągnąć oczekiwaną wydajność aplikacji. Utrudnieniem jest kryterium, gdzie rozpatrywana, oczekiwana wydajność aplikacji również zmienia się dynamicznie.

Artykuł przedstawia także prototyp, którego celem jest adaptacja systemu w taki sposób, aby osiągnąć cel, zdefiniowany jako wydajność aplikacji. Prototyp wykorzystuje zorientowanego na zasoby (*ang. resource-oriented*) wbudowanego w system operacyjny zarządcę obciążeniem (*ang. Workload Manager*), oraz zorientowany na cel (*ang. Goal-oriented*) autorski podsystem bazujący na logice rozmytej. Opisany autorski program gospodaruje zasobami informatycznymi tak, aby jak najlepiej je wykorzystać oraz realizuje translacje zdefiniowanego celu na zajętość wykorzystywanych zasobów systemowych z uwzględnieniem niedeterministycznych współczynników związanych z technologią takich jak czas na alokację i zwalnianie tych zasobów, ich współdzielenie, oraz błędy pomiarów wydajności.

Wyniki badań pokazują, że dzięki zastosowaniu prototypu możliwa jest dynamiczna zmiana celu określonego jako czas odpowiedzi aplikacji a czas znalezienia odpowiednich parametrów mieści się w akceptowalnych granicach mierzonych w sekundach. W artykule przedstawione jest także porównanie przedstawionego prototypu z podobnym, bazującym na koncepcji bazy wiedzy.