Theoretical and Applied Informatics
ISSN 1896–5334
Vol.23 (2011), no. 1
pp. 55–72
DOI: 10.2478/v10179-011-0004-x

Improving efficiency of data-intensive applications in goal-oriented adaptive computer systems

MACIEJ MŁYŃSKI, PRZEMYSŁAW RUMIK

Association for Computing Machinery
{maciej.mlynski},{przemyslaw.rumik}@acm.org

Received 30 December 2010, Revised 17 January 2010, Accepted 14 March 2011

Abstract: An accurate use of the ability to steer computer efficiency is essential from the database point of view. Effective resource allocation is dependent on the performance indicators gathered from running systems. There must be an appropriate balance between accurate measurements, performance indicators and speed of the reallocation algorithms of the computing resources. The extended measurement of efficiency which the authors propose for applications is: the average number of queries within a time unit for particular groups of users. This paper presents an analysis of using the Workload Manager utility in the AIX 5L operating system to improve the efficiency of applications in the MySQL database environment, and an analysis of methods which allows the use of Workload Manager for steering efficiency dynamically.

Keywords: performance management; workload management; dynamic resource allocation; predictive resource management

1. Introduction

Computer systems should be provided with support processes that establish priorities that favour the operating system in terms of the allocation of resources such as processing power, amount of memory used and utilization of I/O channels. This should be done in such a way that processes can perform at a predefined time. [3] Failure of this assumption may reduce the reliability of commercial applications. Failure or lack of efficiency of one element can cause further failure and lack of efficiency of other system components that are related to the first one. This scenario we term a “domino effect”. In specific cases, the "domino effect" might result in lack of functionality of other elements of the system. In the experiments that were carried out, Workload Manager was used. Workload Manager is a tool that allows AIX [7] server computing power allocation according to defined policies.

In order to make better use of system resources, it is necessary to define the characteristics of the class assignment rules for the allocation, and specific system architectures and characteristics of the applications must be considered [4]. Workload Manager is based mainly on conceptual notions such as class assignment rules for classifying resources and processes. The classes are defined as sets of non-defined resources with defined characteristics such as memory occupancy, the use of computing power and server capacity input and output channels [3].

Workload Manager allocates physical resources dynamically, depending on the defined class assignment rules. This gives administrators and architects more control over the scheduling program and virtual memory manager, which in turn allows them to allocate more processing power resources and physical memory in accordance with their needs. In each class, resource limits can be defined, which is helpful for using the application. Limits can be both hard limits, which the application may not exceed, and soft limits, which may be exceeded in exceptional cases.

There at least two kinds of method for resource allocation. The first approach is "goal-oriented" which aims to have previously defined application response times match with their efficiency. The second is "resource-oriented" where only the amount of used resource is defined. How to set the parameters is not defined in the "resource-oriented"approach.

In [5, 9] the authors present work based on a similar environment using Workload Manager, as described in [2]. [5] shows that achieving appropriate automatic settings for such a complex environment is not trivial, and in order to be able to perform proper resource management it needs to be founded on a theoretical basis, which is not yet well defined. [5] also shows an example where increasing resources caused application performance degradation. This was for shared processors working in uncapped modes. In [10] the author shows a prototype for goal-oriented workload management based on a knowledge database. Article [10] also shows that the stage of the learning process for the database is time-consuming, and it could be improved to a limited extent.

In [9] the authors present research and the results of the dynamic resource allocation experiments, based on databases, which were prepared especially for this research. The aim of those experiments was to assess the effectiveness of applying a Workload Manager to keep low the load of distributing system devoted to goal-oriented application. The paper is based on research presented in [9]. With comparison to [9] the following are the main contributions of the paper:

- formal justification of the selection of the used methods,
- shows additional analysis of statistics,
- formalize the problem and draw new conclusions,
- review of the literature in this area.

There are also related works in similar areas such as autonomic computing [1], self-managing systems [21] and control theory. Many methods are already in existence. In [17] and [18] the authors present an adaptive Grey Fuzzy controller. In [15] the MGDC (Goal driven performance controller) is proposed. Closer analysis shows that it is difficult to find a single method for resource allocation that covers so many varied computing environments. In [14] we can see good methods for multisite environments, in [19] for environments based on IA-64, in [20] for Grid computing, and in [16] for virtualized resources through WWW and DB servers using Xen. All the methods presented are different. In [11] the author describes workload characteristics in a production system. A production system is less determinist than computer systems that are available for researchers in laboratories. The workload is very often dependent on external circumstances. A new approach such as virtualization [12, 13] offers a new possibility for changing hardware resources on-the-fly which would have a significant impact on application program characteristics and also resource management, which must be appropriate to the new approach.

This paper presents research using our own testing methods as described in section 2, and the results of the dynamic resource allocation experiments, based on databases, were prepared especially for this research. This database environment, using several applications with different response time expectations, is widely used in actual commercial settings. We used the Workload Manager build in the AIX operating system as an example of a dynamic environment.

2. Load generator

The BaseAttack program (Figure 1) was designed for testing purposes, and also a database was created and configured on the MySQL [6] server. The purpose of the program was, as far as possible, to load the database within the confines of the restrictions of the computing resource defined by Workload Manager.

The program which we designed is as simple as possible, because we wanted to eliminate all effects known to arise from irregular applications. All the experiments had to be repeatable. This program also implemented libraries in order to gather statistics from the program execution and collect them into a readable spreadsheet for future analysis. Our goal was to define and use our own testing methods which are appropriate for the area of investigation. The purpose was to gain greater understanding of Workload Manager behaviour – not application behaviour which is what most testing tools do.

The diagram of the program relies on opening the program, connecting to a database and running a given number of threads so that the existing connection will send queries to the database.

In order to satisfy the conditions of the simulation, in which all the threads tend to

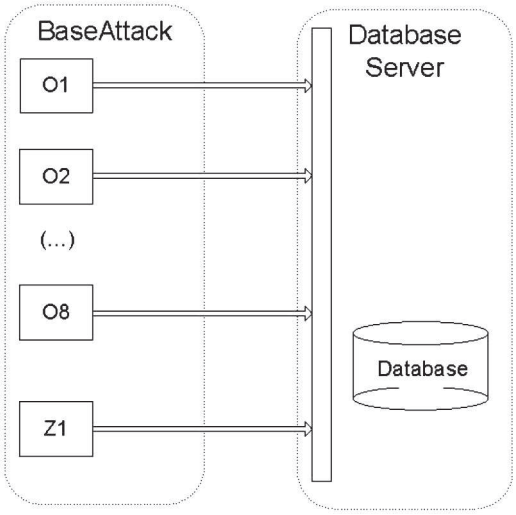

Fig. 1. Logic schema of the program for server load generation

take as much CPU time as possible, this was achieved during the implementation by using an infinite loop. The task threads O1 to O8 were constantly querying the database for the number of records in the POSITIONS array and the maximum value of field X in this table. The task was to thread Z1, adding new records to an array of positions. The metric of the efficiency of the process was the number of queries made by each type of program in the 10-second time segment.

The test database was composed of one table defined as follows:

```
POSITIONS (
ID INTEGER,
X FLOAT,
Y FLOAT,
STAMP TIMESTAMP
);
```

3. Experience in the field of dynamic resource allocation

In order to carry out the experiments, a model case was selected, which is one often found in real computer systems. The current trend is to use Java based multi-user applications connected to a central database system.

On the server used for the experiments, a database server and client applications were installed. In the Workload Manager configuration three classes of resource are defined:

- prod
- prod24
- app

Class prod24 tracked resources used by the MySQL database management system. Class prod and app is the class of clients. During the experiment, measurements were performed in a situation where the system was running nine instances, and a client application was executed by three different users:

- user prod counted in the class with three instances of the program,
- user app counted in the class with three instances of the program,
- user rmk counted in class app with three instances of the program.

The aim of the experiment was to demonstrate the possibility of controlling application performance (measured as the number of queries per unit time) through the choice of system parameters.

The test application named BaseAttack was run using standard compliant JVM of 1.4 supplied by IBM. The MySQL server used during these experiments was version 3.23.58 for IBM-Aix 5.1.0.0 on PowerPC. The BaseAttack application was developed using Java because of the frequent use of the language in business solutions. The program used for connecting to the database was MySQL Connector / J version 3.1.12 developed by MySQL AB. The experiment carried out several tests with different settings of limits in Workload Manager. Table 1 shows the settings used during these experiments.

The aim of each test, based on each set of limits (settings), was to test the impact of the limits of a given group on application efficiency.

During these experiments the above sets were tested, which corresponded to: no limits, limits for the CPU (two sets of tests), limits for memory consumption, and limits for the I/O channel. One experiment was conducted using a set consisting of all the limits. The first test allowed us to say that, in the absence of limits on the complexity of similar requests, all instances of the BaseAttack application achieved a similar performance (Figure 2), and we noted that, in this case, the CPU load is distributed pro rata among all the classes (depending on the number of applications running) (Figures 2 and 3).

In Figure 2, during the first 200 seconds, it was noticeable that the results showed fluctuations in the imposition of the disorder introduced by setting up a Workload Manager mechanism and starting the application. In Figure 2, the Workload Manager settings were the same all the time. However, the efficiency of the application increased

Class	CPU min	CPU max	MEM min	MEM max	I/O min	I/O max
Tests set #1						
prod24	0%	100%	0%	100%	0%	100%
prod	0%	100%	0%	100%	0%	100%
app	0%	100%	0%	100%	0%	100%
Tests set #2a						
prod24	0%	30%	0%	100%	0%	100%
prod	0%	60%	0%	100%	0%	100%
app	0%	10%	0%	100%	0%	100%
Tests set #2b						
prod24	0%	50%	0%	100%	0%	100%
prod	0%	40%	0%	100%	0%	100%
app	0%	10%	0%	100%	0%	100%
Tests set #3						
prod24	0%	100%	0%	60%	0%	100%
prod	0%	100%	0%	35%	0%	100%
app	0%	100%	0%	5%	0%	100%
Tests set #4						
prod24	0%	30%	0%	100%	0%	99%
prod	0%	60%	0%	100%	0%	99%
app	0%	10%	0%	100%	0%	1%
Tests set #5						
prod24	0%	30%	0%	60%	0%	99%
prod	0%	60%	0%	35%	0%	99%
app	0%	10%	0%	5%	0%	1%

Tab 1. Limits for the Workload Manager, used during these experiments

	app [queries per 10 seconds]	**prod [queries per 10 seconds]**	**rmk [queries per 10 seconds]**
getMaxX	95 (113)	98 (118)	103 (117)
getCount	95 (112)	98 (119)	102 (117)
putPosition	21 (22)	15 (17)	18 (17)

Tab 2. Summary of performance for requests getMaxV and getCount 999, and 171 putPosition requests

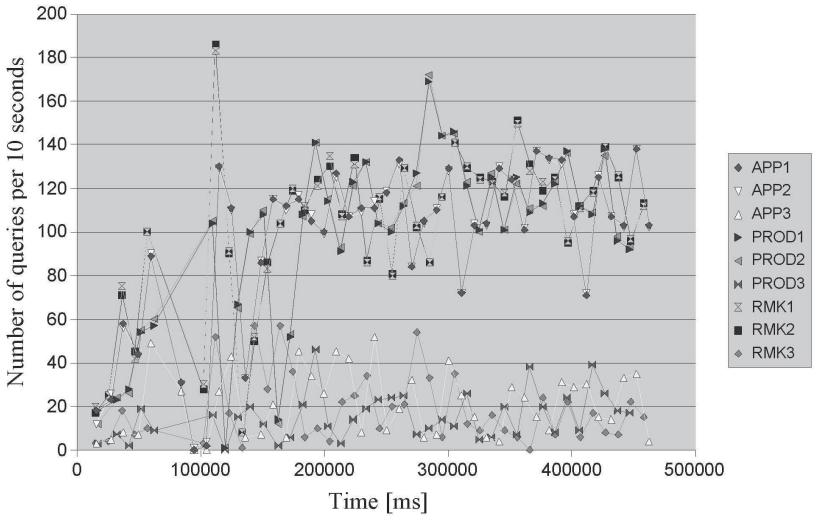

Fig. 2. BaseAttack application efficiency without limits (results related to Tab. 2)

slightly. This is the result of caching of data, and database memory optimization mechanisms. During the experiments the computer system was fully utilized. Besides that, the processors were fully utilized – there were several threads ready to run in queue to those processors. In Figure 2, there are sometimes anomalies such as peaking at more than 180 queries for around 100 seconds. This had happened because of the high processor utilization.

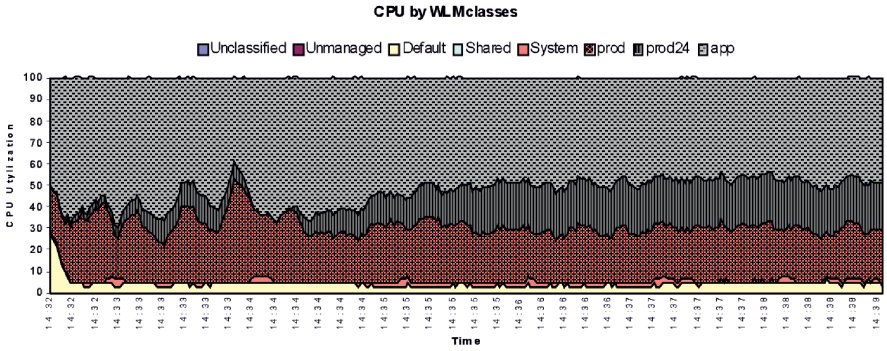

Fig. 3. Processor utilization for particular classes without limits (tests set #1)

Another hypothesis is that all the other processes were blocked and then the two processes from class RMK (RMK1 and RMK2) had too much processing power available and could process more than 180 queries. Figure 5 shows the second test which

consisted of imposing limits on CPU usage for classes. In these classes, client applications were run on the database server. Tests with the imposition of limits on CPU usage reveal that such limits have a significant impact on application efficiency in a working class, which imposed the limits. The phenomenon is that, despite the imposition of a greater limit on CPU usage for the prodclass (60% for set # 2, and 40% for set # 2b), applications running in this class showed higher productivity. Because lowering the limits for Class prod was associated with an increase in the limit for class prod24 (from 30% to 60%), which runs the database server, this was a reasonable explanation. The lower limits for the class made it impossible to work the database server efficiently, as defined in resource class prod24.

Such behaviour of the server and client applications indicates that the real problem appears when clients seize the CPU time which should be allocated to the database server.

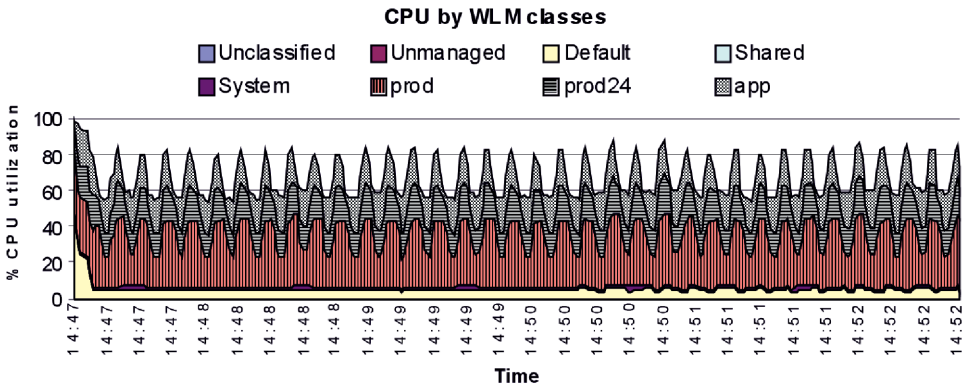

Fig. 4. Processor utilization with limits for the processors' use (tests set #2a)

This shows a comparison of a summary of efficiency (i.e. the total number of queries within 10 seconds) for a system with no limits (set # 1), such that the aggregated performance of 999 getMaxX queries, 999 getCount queries and 171 putPosition queries is significantly lower than for a system with applied limits (set # 2b) (Figure 6 and 7) where the aggregated numbers of getMaxX and getCount queries is equal to 1248, and putPosition queries are equal to 219 per 10 seconds.

In Figure 4, we see the wave that is caused by the internal algorithm implemented in Workload Manager. In Workload Manager implemented in the AIX 5L operating system, the decision system is implemented based on class colouring [3]. Each class have a colour assigned to it, which changes depending on the usage of computing resources. When the class is using no resources, or much fewer than it should, the colour is cold (blue), which means that when new resources come to the operating system, and the assigned class rules decide in which resource class the new processes should

be computed, then the operating system can decide quickly that the resources may be computed. When the resource class is using resources near or above the defined limits, then the class colour is warm (red), which indicates to the operating system kernel that the new resources should wait and can only be allocated when the class is colder.

The only method which the operating system has available to disinherit or not allocate resources is to decrease process priority. Then, when the priority is low, the processes use fewer computing resources. This works fine for the CPU, whereas for memory allocation a lower priority for processing does not always means that the process will release the earlier allocated memory quickly.

In Figure 4, we can see how the above described algorithm works, and the effect of changing the process priorities.

	app [queries per 10 seconds]	**prod [queries per 10 seconds]**	**rmk [queries per 10 seconds]**
getMaxX	47	182	49
getCount	47	182	49
putPosition	10	32	11

Tab 3. Global efficiency for 834 getMaxX and getCount requests, and for 159 putPosition requests

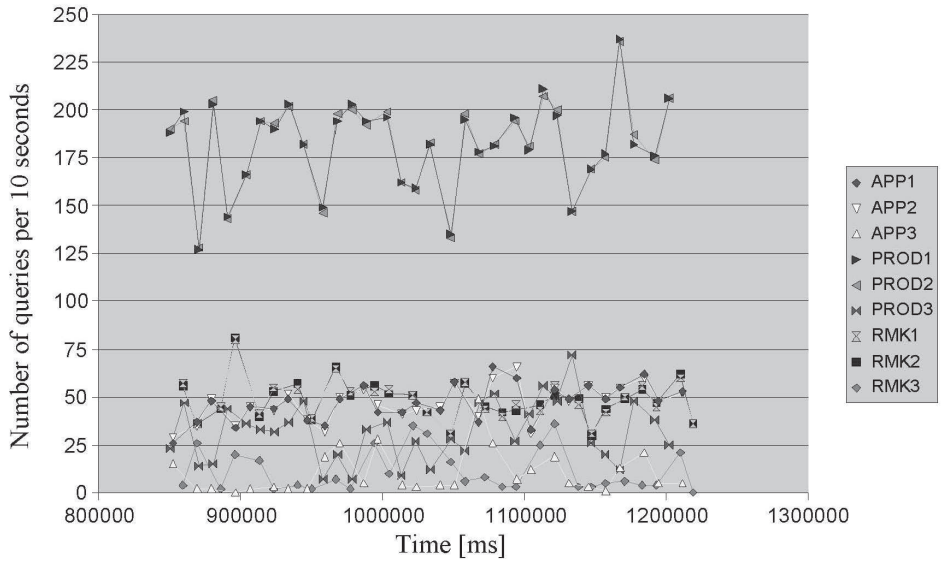

Fig. 5. Efficiency (defined as numbers of requests to choose instances executed by particular users) (tests set #2a) (results related to Table 3)

	app [queries per 10 seconds]	prod [requests per 10 seconds]	rmk [requests per 10 seconds]
getMaxX	52	314	50
getCount	52	314	50
putPosition	12	51	10

Tab 4. Global efficiency for 1248 getMaxX and getCount requests, and 219 putPosition requests

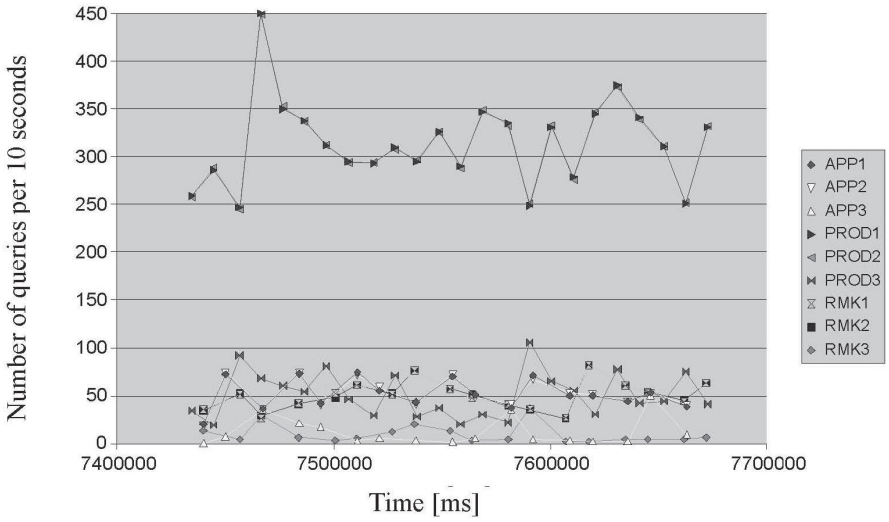

Fig. 6. Efficiency (defined as numbers of requests to choose instances executed by particular users) (tests set #2b) (results related to Table 4)

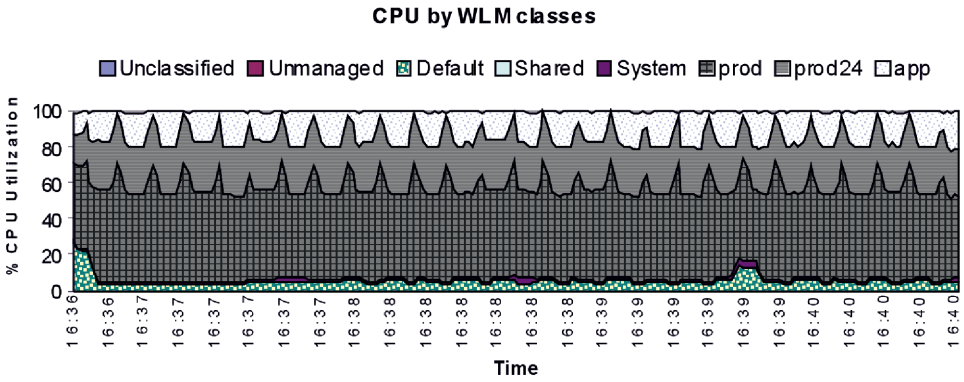

Fig. 7. Processor utilization for particular classes with WLM limits for processors (tests set #2b)

Another group of tests sought to examine the impact of limits on the amount of memory (set # 3) and on the I/O channel (set # 4). Graphs show how memory consumption (Figure 8) with the limits set by # 3, and the breakdown of memory between the different classes, was relatively stable despite the limits imposed, by suggesting that the limits were higher than the requirements of the applications working in their respective classes.

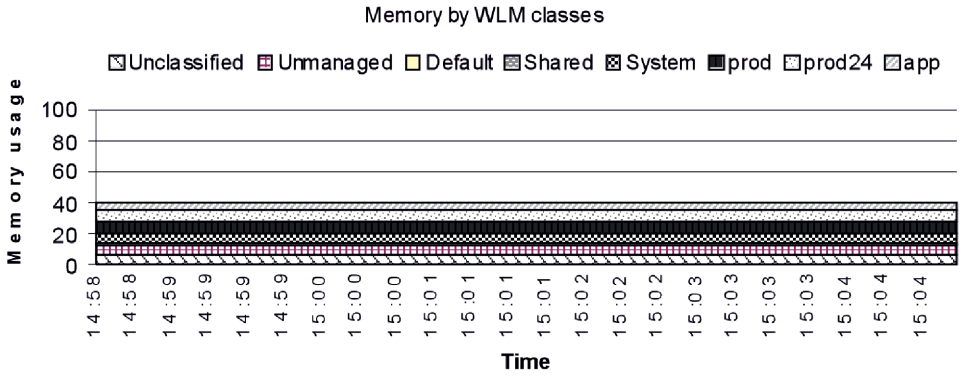

Fig. 8. Utilization of memory with limits on its occupation (tests set #3)

The test system with limits imposed on the amount of memory shows no change in performance compared with a system without limits (see Figure 9 for the limits on the amount of memory), suggesting that, in the case under examination, there is no clear relationship between performance and memory limits.

	app [queries per 10 seconds]	**prod [queries per 10 seconds]**	**rmk [queries per 10 seconds]**
getMaxX	111	118	121
getCount	111	118	121
putPosition	24	25	25

Tab 5. Global performance for 1050 getMaxX and getCount requests,and 222 putPosition requests (tests set #3)

Also, imposing limits on the scheme for the I/O channel had no significant impact on productivity, which is comparable to that for a system without imposed limits (Figure 10).

The intention of the final experiment was to observe a series of system behaviours with the imposition of limits on CPU time, as well as the amount of memory and I/O channels. Previous experience suggests that the imposition of any limits (set # 5) should give similar performance to that of a setup of limits only on CPU time (set # 2).

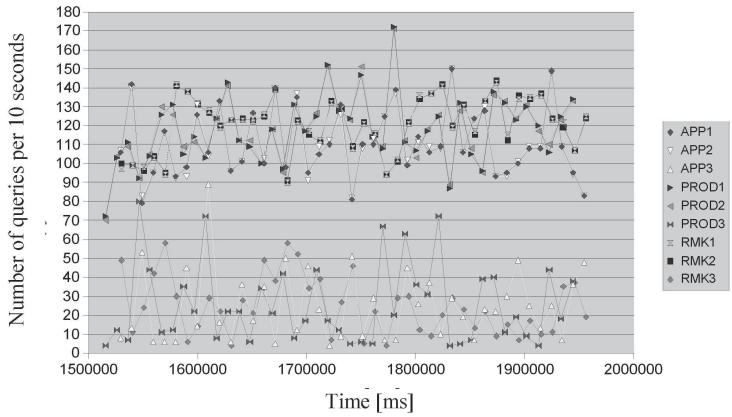

Fig. 9. Efficiency (defined as numbers of requests to choose instances executed by particular users) (tests set #3) (results related to Table 5)

	app [queries per 10 seconds]	**prod [queries per 10 seconds]**	**rmk [queries per 10 seconds]**
getMaxX	116	112	117
getCount	116	112	117
putPosition	25	28	29

Tab 6. Global performance for 1035 getMaxX and getCount requests and 245 putPosition requests (tests set #4)

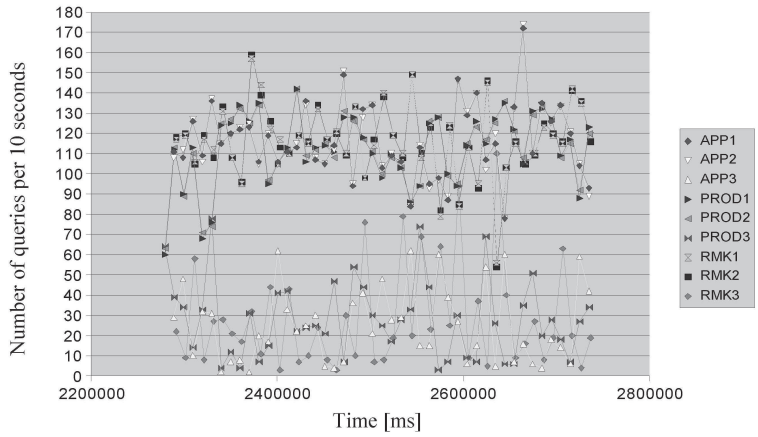

Fig. 10. Efficiency (defined as numbers of requests to choose instances executed by particular users) (tests set #4) (results related to Table 6)

However, it appears that the imposition of limits set by # 5 resulted in a significant decrease in performance compared to a system without imposed limits. A deeper analysis suggests that the average yield for this case is about 20% higher than recorded, but the measure imposes an additional load on the server.

	app [queries per 10 seconds]	**prod [queries per 10 seconds]**	**rmk [queries per 10 seconds]**
getMaxX	20	156	28
getCount	20	156	28
putPosition	2	34	3

Tab 7. Global performance for 612 getMaxX and getCount requests and 117 putPosition requests (tests set #5)

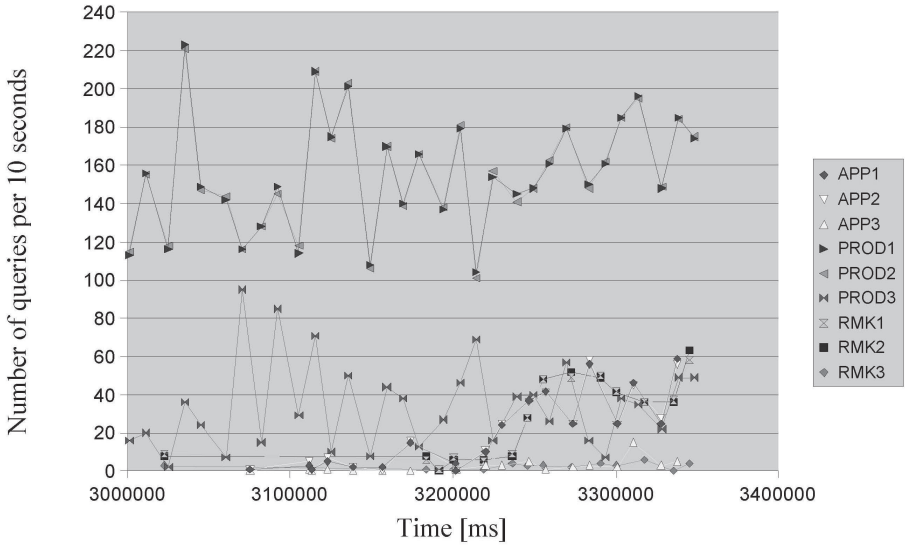

Fig. 11. Efficiency (defined as numbers of requests to choose instances executed by particular users) (tests set #5) (results related to Table 7)

Results obtained for the tested limits suggest that it is possible to use Workload Manager (or its equivalent) to improve the performance of the application and database server.

4. Analysis of the statistics

Selecting the appropriate limits for the class, in which there is a database server process running, allows avoidance of a race (threshing) between the client application

and the server for access to CPU time. Mechanisms provided by Workload Manager, enriched with simple counting mechanisms in applications, allow us to determine the safe limits for the class-server by carrying out measurements for running Workload Manager without limits and running a database process and applications processes.

Execution in the system of the application (in class prod) by the prod user, and simultaneously MySQL engine (in class prod24) without limits (Figure 12), helps us to determine the class-limits required by the server. A similar test also allows us determine the relationship between the limits applied in Workload Manager and application performance.

The required value of the limits for class prod24 in the examined model was 25%.

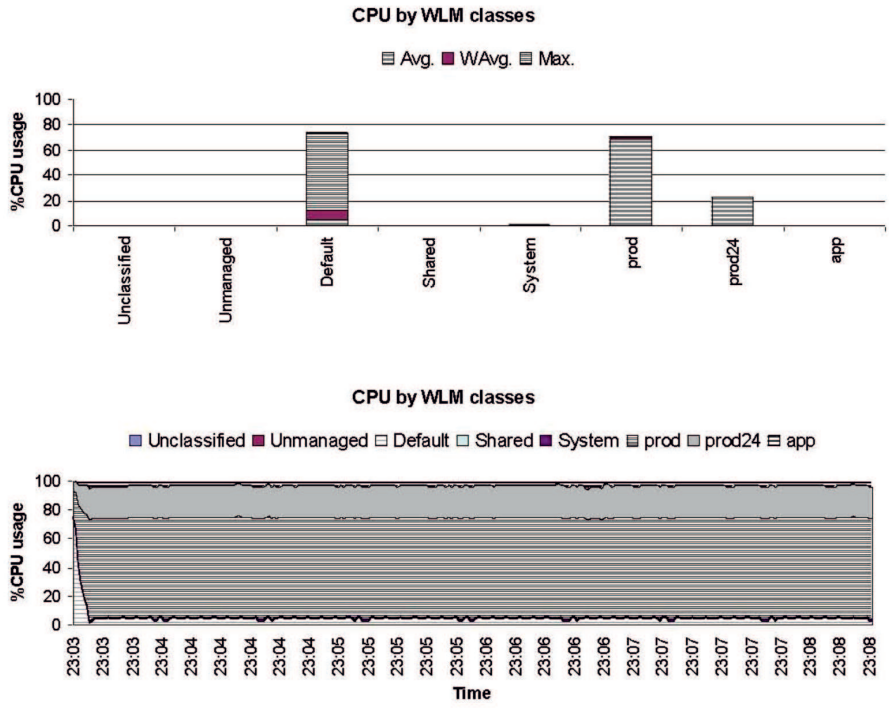

Fig. 12. Processor load caused by class prod (client application) and class prod24 (server MySQL)

In the examined example, there was only a correlation between the CPU limit and the application performance. This is described by equation 1. In the presented research, for primary performance metrics, the efficiency of getMaxX and getCount queries was used.

$$CPU_{thread}\% = \frac{CPU_{MAX}}{efficiency_{\max}} \cdot \exp ected\_efficiency \tag{1}$$

Knowing the value of the limits of the CPU for a single application instance and the number of applications running within a class can be the designated limit for a class with the authors' assumption that each application within the class uses the CPU to the same extent (the assumption we adopt on the basis of the earlier measurements).

5. Conclusions

In applications where the emphasis is on their service quality, defined as the ability to process a particular number of requests, predictive workload management is essential. This feature can therefore improve the operation of conventional mechanisms for dynamic allocation of resources in an operating system.

The result of the presented experiments shows that, after applying appropriate limits (settings) for Workload Manager, the efficiency of the resource classes for production more than doubled. The tests carried out have shown that it is not a trivial matter to choose the parameters of Workload Manager. Several ad-hoc settings were applied. The same settings did not always result in the same application efficiency in the database environment. Empty databases behave differently than databases that have collected millions of records.

The experiments presented in this paper show that it is better to use soft limits than hard limits. Soft limits allow one to use all the resources from the server. There are many cases where hard limits would be more suitable. Hard limits are better for real-time applications, or applications where short processing time is unnecessary.

In [10] the author described a resource management system based on a knowledge database. The learning process for building this knowledge database was automatic and permutations of Workload Manager settings were used. In each permutation the application efficiency was measured. In the experiments presented here, the ad-hoc settings were based on the authors' knowledge of how their application works. The application used in these experiments was written especially for this occasion. We decided to build our own testing methods so as to be sure that the experiments are repeatable. In complex and irregular applications the knowledge of how to set up accurate Workload Manager settings to gain expected response times is more difficult. The results of the experiments as presented also show that, besides a theoretical background, knowledge about operating system behaviours is also important. Even when using a simple (regular) application and constant conditions, the measurements are different. This is presented in Figures 2, 5, 6, 9, and 10. Further analysis shows that the same request, for instance reading the same data from the same table, might takes different amounts of time, due to caching, running background processes, or just memory allocation into paging spaces.

The program and the testing methods used in these experiments are not relevant for irregular applications. Irregular applications are much more difficult to manage. The

testing methods used in this paper can only be used in part for this purpose. Irregular applications need more complex and time-consuming testing strategies. This work can be continued towards building a machine that will perform a series of tests and look for optimal performance. Perhaps genetic algorithms [8] or other more sophisticated methods can be used for this purpose.

References

1. L. W. Russell, S. P. Morgan, E. G. Chron: *Clockwork: A new movement in autonomic systems*, IBM Systems Journal, pp. 77-84, 2003.
2. S. Castro, N. Tezulas, B. Yu, J. Berg, H. Kin, D. Gfroerer: *AIX 5L Workload Manager (WLM)*, IBM International Technical Support Organization, Austin 2001.
3. M. Mlynski: *Dynamic resource allocation in AIX 5L*, In 12th Conference of Real Time Systems, WKL, pp. 247-256, 2005.
4. M. Mlynski: *Analysis of using an AIX dynamic resource allocation mechanism to describe a utility level of servers in an Oracle database environment*, Studia Informatica (formerly Zeszyty Naukowe Politechniki Slaskiej), Vol. 26, no. 3 (64), Gliwice 2005.
5. M. Mlynski: *The influence of the IBM pSeries servers virtualization mechanism on dynamic resource allocation in AIX 5L*, Scalable Computing, Practice and Experience Scientific International Journal for Parallel and Distributed Computing, Volume 10, no. 2, pages 189–199, June 2009.
6. S. K. Cabral, K. Murphy: *MySQL Administrator's Bible*, Willey, 2009.
7. N. Tickett, T. Nakagawa, R. Mani, D. Gfoerer: *Understanding IBM @server pSeries Performance and Sizing,* IBM International Support Organization, Austin 2001.
8. Y. Long, J. Connan, *KernTune: self-tuning Linux kernel performance using support vector machines*, In Proceedings of the 2007 annual research conference of the South African Institute of Computer Scientists and on IT research in developing countries, 2007.
9. M. Mlynski, P. Rumik: *Examination of usefulness of Workload Manager in AIX 5L to improve efficiency of applications in an MySQL database*, In 2nd Conference of Database, Applications and Systems, WKL, Warsaw 2006.
10. M. Mlynski: *Automatic Adjustment of the settings of Workload Manager for adaptive performance management*, Theoretical and Applied Informatics, Vol. 21, no. 1, pp. 37-57, 2009.
11. M. Mlynski: *Analysis of using an AIX dynamic resource allocation mechanism to describe a utility level of server in an Oracle databases environment*, Studia Informatica (formerly Zeszyty Naukowe Politechniki Slaskiej), Vol. 26, no. 3 (64), 2005.
12. M. Mlynski: *Understanding performance metrics and their collection in dynamic virtual machines*, Theoretical and Applied Informatics, Vol. 22, no. 2, pp. 115-130, 2010.

13. C. Matthys, M. Mlynski, N. Tollet, G. Barbati, H. Chauhan, B. Dierberger, R. Marchini, H. Wittmann: *Planning, Installing and Using the IBM Virtualization Engine Versin 2.1*, IBM International Technical Support Organization, Poughkepsie (USA), 2006.
14. M. Srivatsa, N. Rajamani, M. Devarakonda: *A Policy Evaluation Tool for Multiside Resource Management*, IEEE Transactions on Parallel and Distributed Systems, Vol. 10, no. 10, 2008.
15. J. Aman, C. K. Eilert, D. Emmes, P. Yocon, D. Dillenberg: *Adaptive algorithms for managing a distributed data processing workload*, IBM Systems Journal, Vol. 36. no. 2, pp. 242, 1997.
16. P. Padala, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, K. Salem: *Adaptive Control of Virtualized Resources in Utility Computing Environment*, Proceedings of the 2nd ACM SIGOPIS/EuroSys European Conference on Computer Systems, 2007.
17. E. Kayacan, O. Kaynak: *An Adaptive Grey Fuzzy PID Controller With Variable Prediction Horizon*, In Proceedings of SCIS&ISIS 2006, Japan, 2006.
18. X. Xianghua, Y. Yanna, W. Jian: *Gray Prediction Control of Adaptive Resources Allocation in Virtualized Computing System*, Eight IEEE International Conference on Dependable, Autonomic and Secure Computing, 2009.
19. S. H. 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, pp. 1642-1656, 2008.
20. Y. Wu, K. Hwang, Y. Yuan, W. Zheng: *Adaptive Workload Prediction of Grid Performance in Confidence Windows*, IEEE Transactions on Parallel and Distributed Systems, 2009.
21. Y. Diago, J. L. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, D. Phung: *Self-Managing Systems: A Control Theory Foundation*, In 1st Workshop on Operating System and Architectural Support on Demand IT Infrastructure, 2004.

Poprawa wydajności aplikacji intensywnie wykorzystujących dane w zorientowanych na cele adaptacyjnych systemach komputerowych

Streszczenie

Właściwe wykorzystanie zdolności do sterowania wydajnością systemu komputerowego jest niezbędna z punktu widzenia odbiorcy usług informatycznych. Efektywna alokacja zasobów obliczeniowych jest uzależniona od zebranych metryk wydajności. Należy, więc zachować właściwą równowagę pomiędzy dokładnością pomiarów, oraz szybkością algorytmów służących do realokacji zasobów obliczeniowych rozważanego systemu komputerowego.

Proponowany przez autorów rozszerzony pomiar efektywności dla aplikacji to średnia liczba zapytań w jednostce czasu dla poszczególnych grup użytkowników. Taka

metryka jest celem do zrealizowania w badanym systemie komputerowym. W artykule przedstawiono analizę wykorzystania zarządcy obciążeniem w systemie operacyjnym AIX 5L do poprawy wydajności aplikacji w środowisku bazy danych MySQL. Zaprezentowano również analizę metod, które pozwalają na korzystanie z zarządcy obciążeniem do dynamicznego sterowania wydajnością.

Autorzy analizują zachowanie się systemów nieregularnych. Takie systemy charakteryzują się dość wysokim niedeterminizmem, objawia się to tym, że wielokrotne wykonanie pomiaru obciążenia przy jednakowych warunkach, może dać różne rezultaty. Takie zachowanie systemu jest powodowane magazynowaniem danych w podręcznych strukturach pamięci oraz działaniem systemowych algorytmów przydzielania i zwalniania zasobów informatycznych.

Autorzy wykorzystują do badań samodzielnie przygotowane programy i procedury testujące. Program *BaseAttack* napisany został w języku Java, co sprawia, że testowane środowisko jest zbliżone do stosowanych obecnie w przedsiębiorstwach nowoczesnych systemów komputerowych. Testowane środowisko zostało podzielone na podklasy. Procesy systemowe o większym znaczeniu dla użytkownika są w innych klasach niż procesy mniej znaczące. Wyniki eksperymentów pokazują, że możliwe było zwiększenie wydajności wybranych podklas niemal dwukrotnie bez ingerencji w ustawienia wewnętrznych parametrów bazy danych.