Scalable Workload Adaptation for Mixed Workload

Baoning Niu and Jian Shi

Taiyuan University of Technology
79 West Yingze Street
Taiyuan, Shanxi, China
niubaoning@tyut.edu.cn, shijianzhengzhou@126.com

Abstract. Workload adaptation is a performance management process in which an autonomic database management system (DBMS) efficiently makes use of its resources by filtering or controlling the workload presented to it in order to meet its Service Level Objectives (SLOs). The overhead incurred by filtering or controlling the workload is an important factor affecting the effectiveness of workload adaptation. This paper investigates the overhead of AWMF, a framework for workload adaptation and proposes a scalable approach for adapting mixed workload under the framework. The proposed approach allows Query Scheduler, the prototype implementation of AWMF, manage both OLAP and OLTP classes of queries to meet their performance goals by allocating DBMS resources through admission control in the presence of workload fluctuation. Experiments with IBM® DB2® Universal Database™ are conducted to show the proposed approach is scalable and effective.

Keywords: Workload adaptation, Performance management, DBMSs.

1 Introduction

One of the challenging problems that contemporary database management systems (DBMSs) are facing is to guarantee the service level objectives (SLOs) for their complex workloads. The emerging trend of server consolidation results in a workload with diverse and dynamic resource demands with competing performance objectives for different applications. Web-based applications introduce a need for flexible and guaranteed application service levels [1]. In order to meet the SLOs, DBMSs must be able to aware the workload changes and allocate resources accordingly.

Workload adaptation is such a technique to solve the problem. It can be defined as a performance management process in which an autonomic DBMS efficiently makes use of its resources by filtering or controlling the workload presented to it in order to meet its SLOs [6]. A general framework for workload adaptation in autonomic DBMSs, called AWMF, has been proposed in [6]. The framework consists of a workload detection process and a workload control process, involving four functional components, namely workload characterization, performance modeling, workload control and system monitoring. The prototype implementation of the framework, called Query Scheduler, using cost-based workload control is proven to be effective for OLAP workload. Workload control based on multiprogramming levels (MPL) for

OLTP workload under similar framework was discussed in [8]. The performance controller sits at the front of DBMSs, intercepts queries and makes admission control.

Mixed workloads with both OLAP and OLTP queries are common. Controlling mixed workloads is more complex than homogenous workloads. Control of OLAP workloads based on cost is appropriate because the size of OLAP queries varies widely. Since OLTP queries are small, control of OLTP workloads based on MPLs can eliminate the overhead to acquire costs of queries. However, the overhead from a separate controller is still significant to the OLTP queries with sub-second execution time. Usually a transaction consists of several dozens of queries. The accumulative delay from the controller could be several times more than its execution time.

This paper investigates the overhead incurred by controlling workloads in AWMF and explores the techniques can be used under AWMF for a mixed workload using cost-based workload control. The rest of the paper is structured as follows. Section 2 briefly describes AWMF and its prototype implementation – Query Scheduler, for workload adaptation in autonomic DBMSs. Section 3 analyzes the overhead of Query Scheduler and related scalable issues. Section 4 discusses the scalable approach for handling mixed workload in Query Scheduler. The evaluation of the proposed approach for mixed workload is outlined in Section 5. We conclude and suggest future work in Section 6.

2 AWMF and Its Prototype Implementantion

In this section, we briefly introduce AWMF and its prototype implementation for subsequent discussion. For detailed description, please refer to [6].

2.1 AWMF

Workload adaptation is a process of optimizing resource usage by controlling the workload presented to the system. As shown in Fig. 1, the framework for workload adaptation consists of two processes, namely workload detection and workload control, involving four functional components: workload characterization, performance modeling, workload control, and system monitoring.

Workload characterization is concerned with measuring and modeling production workloads [4, 5]. The purpose of characterizing a workload is to understand and determine the resource usage and performance behavior for subsequent workload control. Performance modeling tries to predict the performance of the target system through a model that describes the features of the target system [5]. Workload control components find and enforce an optimal workload control plan to meet the performance objectives when fluctuation in the workload causes the system performance to degrade. System monitoring, or feedback, indicates how well the system is performing by continuously acquiring the execution information of the workload and the resource usage of the system.

Workload detection identifies workload changes by monitoring and characterizing current workloads and predicting future workload trends. As shown in Fig. 1, two functional components, workload characterization and system monitoring, are involved in the workload detection process.

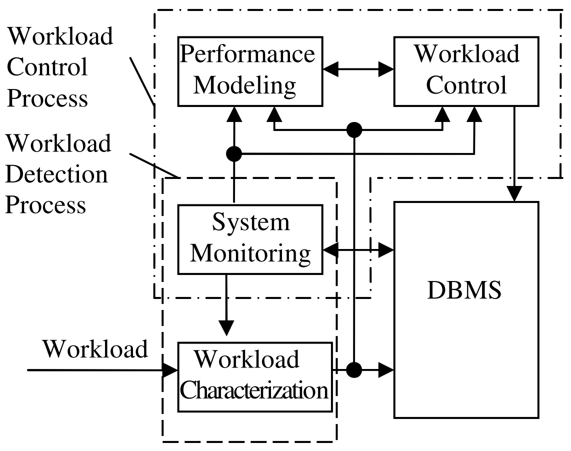

Fig. 1. AWMF

Workload control involves system management via efficient allocation of resources. One of the main issues regarding workload control is how to determine the appropriate amount of control. Under the framework, when workload changes are detected, the workload control component determines whether or not an adjustment is needed. In the positive case, it generates workload control plans and submits them to the performance modeling component for evaluation. It then chooses the optimal plan to exert control over the workload. Three functional components, the workload control, the performance modeling and the system monitoring, are involved in the workload control process.

2.2 Query Scheduler

The Query Scheduler shown in Fig. 2 is a prototype implementation of the workload adaptation framework with IBM® DB2® Universal Database system (DB2 UDB).

Query Scheduler uses Query Patroller [3] (DB2 QP), the workload controller in DB2 UDB, to intercept queries and acquire query information and, via direct commands to DB2 QP, release queries. In this implementation, DB2 QP is configured to automatically intercept all queries, record detailed query information, block the DB2 agent responsible for executing the query until an explicit operator command is received. Finally, DB2 QP was modified to inform Query Scheduler each time a query was intercepted. The Monitor then collects the information about the query from the DB2 QP control tables, including query identification information, query cost, query execution information etc. The Monitor passes the query information to the classifier and the scheduling planner. The Classifier assigns the query to an appropriate service class based on its performance goal and places the query in the associated queue manipulated by the dispatcher. The Dispatcher receives a scheduling plan from the Scheduling Planner and releases the queries in the class queues according to the plan.

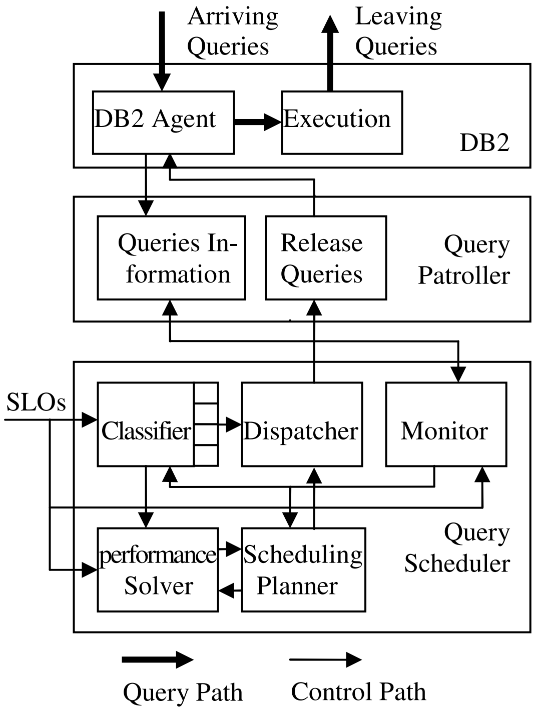

Fig. 2. Query Scheduler

Since workload control is cost-based, a scheduling plan is a set of class cost limits. Each service class is assigned a class cost limit expressed in *timerons*, which is a generic cost measure used by the DB2 UDB optimizer to express the combined resource cost to execute a query. This limit is the maximum allowable total cost of all concurrent queries belonging to a service class. The Dispatcher releases a query for execution by calling the unblocking API provided by DB2 QP, which releases the blocked agents. The Scheduling Planner consults with the Performance Solver at regular intervals to determine an optimal scheduling plan, and passes this plan to the Dispatcher.

3 Overhead of Query Scheduler

The overhead incurred by Query Scheduler includes query interception, acquisition of query information, control logic, and query release as shown in Fig. 3. DB2 provides query information to DB2 QP through TCP/IP connections at the points where a query arrives or terminates. DB2 QP maintains the TRACK_QUERY_ INFORMATION table [3] to store the query information. When DB2 QP writes to the table, a trigger on the table is activated and sets up a TCP socket to inform Query Scheduler a query is intercepted or terminated. In either case, Query Scheduler requests the query information through the socket. There are four times of TCP / IP communication, two writes and two reads for DB2 QP. Query Scheduler acquires query information, makes a control decision to release a query at the two points where a query is intercepted and terminated. As we can see the overhead for Query Scheduler is mainly from DB2 QP, especially the two writes and two reads when compared to the other activities.

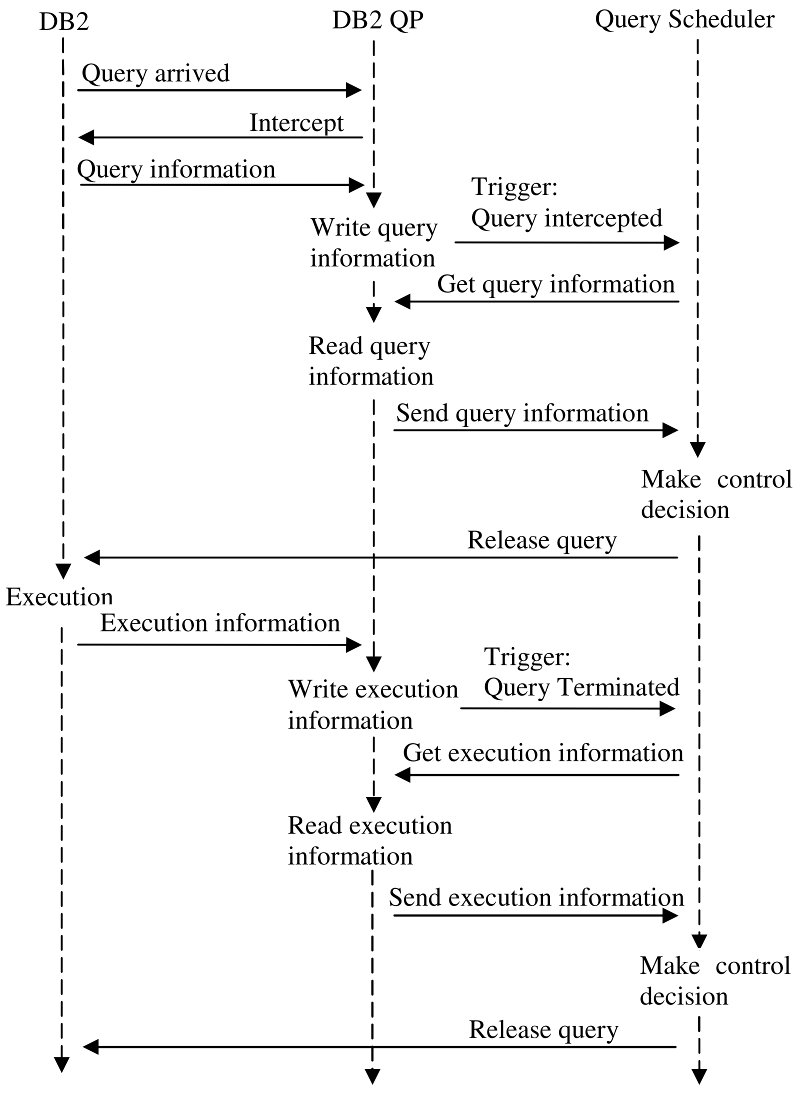

Fig. 3. Overhead of Query Scheduler

We measured the throughputs for OLAP and OLTP workloads with DB2 QP turned on and off. When DB2 QP is turned on, a predetermined total cost limit is set to keep the system from overloading. The OLAP workload is composed of 22 TPC-H [9] queries and the OLTP workload consists of 5 types of transactions from TPC-C [9] benchmark. As shown in Fig. 4, when DB2 QP is turned off, the throughput of the OLAP workload is smaller than when DB2 QP is turned on. This is because the benefit from DB2 QP protecting the system from overload is much bigger than the delay from DB2 QP. Besides, the delay is negligible compared with the long execution time of the OLAP queries.

On the other hand, as shown in Fig. 5, when DB2 QP is turned off, the throughput of the OLTP workload is much higher than when DB2 QP is turned on. This suggests that the delay from DB2 QP is significant to the OLTP queries. The delay from DB2 QP is inherent unless the Query Scheduler is implemented in the kernel of DBMSs, which would eliminate the overhead shown in Fig. 3.

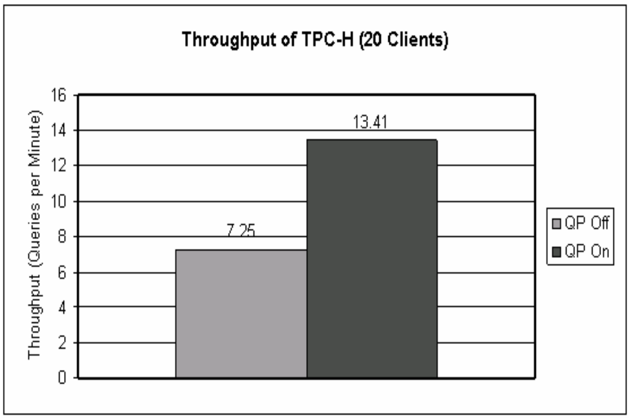

Fig. 4. The overhead of Controlling OLAP Workload with DB2 QP

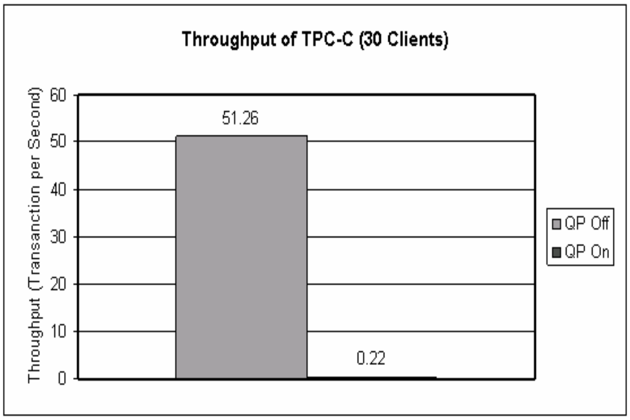

Fig. 5. The overhead of Controlling OLTP Workload with DB2 QP

4 The Scalable Approach

Given the overhead significantly outweighs the sub-second execution time of the OLTP queries, we need to find approaches, besides direct DB2 QP control, to adapt the OLTP queries. With a mixed workload, if we assume that OLTP queries are assigned the highest importance level, which is generally the case in production workloads, then we can indirectly control OLTP queries by controlling the competing OLAP classes. Query Scheduler can allocate more resources to OLTP queries by lowering the cost limit of competing OLAP class and can decrease resources allocated to OLTP queries by raising the cost limit of the competing OLAP class. To do so, we configure DB2 QP to intercept OLAP queries and bypass OLTP queries.

The rest of this section discusses the detailed implementation issues of the scalable approach for the mixed workload with Query Scheduler.

4.1 Choosing Performance Metrics

The most widely used performance metrics are response time, throughput, and execution velocity [2]. Response time and throughput are appropriate when the queries are similar in size. For workloads with widely varying response times, execution velocity, which is a measure of the time a query spends executing compared to its total time in the system, is a better choice. In this implementation, we choose the average response time for OLTP workload and query velocity for OLAP workload. Query Velocity is defined as

$$Query\_Velocity = Execution\_Time \,/\, Response\_Time \; . \qquad (1)$$

For average response time goals, a smaller value means better performance. For query velocity goals, a larger value indicates better performance.

4.2 Performance Modeling

For the OLAP workload class, the performance model is [6]:

$$V^k = V^{k-1} C^k / C^{k-1} \; , \qquad (2)$$

where V^{k-1} and V^k are the performance of the OLAP class at the $(k-1)^{th}$ and k^{th} control interval respectively. C^{k-1} and C^k are the class cost limit of the OLAP class at the $(k-1)^{th}$ and k^{th} control interval respectively.

For the OLTP class, we cannot simply use this model. First, the performance metric is different. The OLAP class use query velocity, while the OLTP class uses average response time. Second, the system cannot control the OLTP class directly and the total cost of OLTP queries in the system is unknown. Third, OLAP queries tend to be I/O intensive and OLTP queries are CPU intensive. The cost limit for the OLAP class is not directly usable for the OLTP class. We measured the average response time of the OLTP class relative to the cost limit of the OLAP class (Fig. 6). The number pairs in the legend are (number of OLTP clients, number of OLAP clients). The average response time of the OLTP class is almost linear increases with the increase of the cost limit of the OLAP class when the system is not overloaded (system total cost limit less than 300K timerons). Based on this knowledge, we use a linear model to model the performance of the OLTP workload.

$$t^k = t^{k-1} + s(C^k - C^{k-1}), \tag{3}$$

where t^{k-1} and t^k are the average response time of the OLTP class at the $(k-1)^{th}$ and k^{th} control interval respectively. C^{k-1} and C^k are the class cost limit of the OLAP class at the $(k-1)^{th}$ and k^{th} control interval respectively. s is a constant and can be obtained by using linear regression.

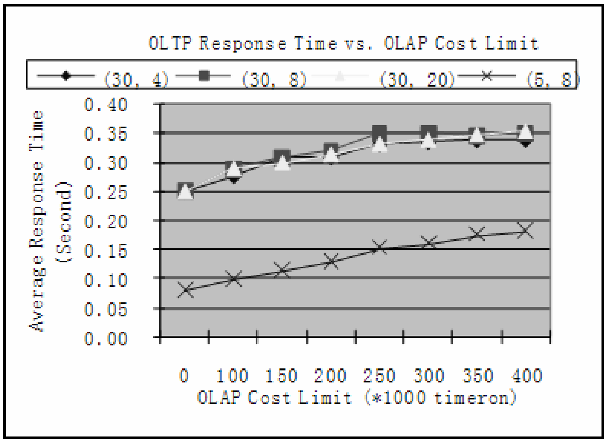

Fig. 6. OLTP performance vs. OLAP cost limit

4.3 Utility Functions

Utility functions are the key to encapsulating multiple SLOs into an objective function and play an important role in allocating resources among workload classes. There is no single ways to choose utility functions. The principles to choose utility functions are:

- The utility should be larger when the performance experienced by a service class is better than the performance goal and smaller when the performance is worse;
- The utility should increase as the performance experienced by a service class increases and decrease otherwise;
- The speed of utility increase should be getting slower with the increase of performance and the speed of utility decrease should be getting faster with the decrease of performance;
- The size and shape of the utility function should be controlled by one or two parameters that can be adjusted by the platform provider to reflect the importance of one class of traffic over another [7].

Even with these principles, choosing a set of proper utility functions is not easy. Based on our experiments we found a general form of utility functions:

$$u = 1 - e^{aI\frac{\bar{g}-g}{g-g_w}} , \tag{4}$$

where $\bar{g}$ is the performance goal of the service class to be achieved. I is the importance level of the service class. g_w is the worst performance allowed. g is the real performance. a is a constant that can be assigned by users to reflect the extent of difference between two adjacent importance levels.

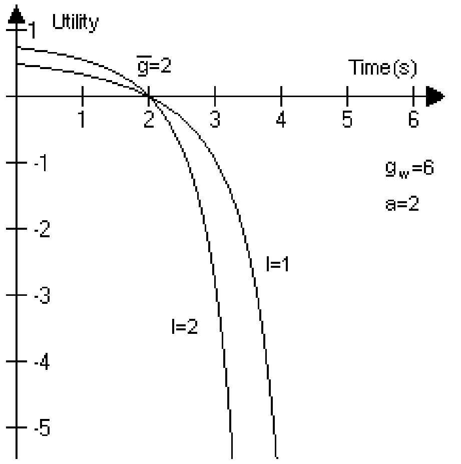

Fig. 7. Utility Function for Response Time Goals

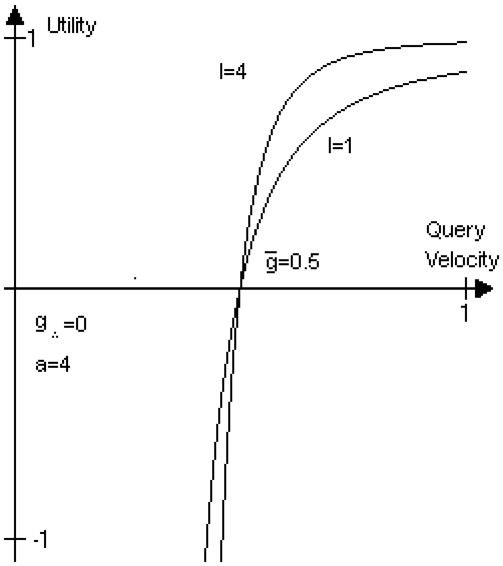

Fig. 8. Utility Function for Query Velocity Goals

The utility function not only complies with the principles and encapsulates both the performance goal and the importance level, but also has good properties (the second derivative exists) allow mathematic methods to be used to optimize the objective functions. Besides, it can be applied to both response time and execution velocity. Fig. 7 shows the utility function applied to response time goals and Fig. 8 for query velocity goals.

5 Experiments

In this section we evaluate the proposed approach to adapting mixed workload with a set of experiments.

The computer system used as the database server is an IBM xSe-ries® 240 machine with dual 1 GHZ CPUs, four PCI/ISA control-lers, and 17 Seagate ST 318436LC SCSI disks. We use IBM DB2 UDB Version 8.2 and Query Patroller as supporting components.

The system cost threshold is determined in the same way in [6].

5.1 Workload

We use the TPC-H benchmark as our OLAP workload and TPC-C benchmark as our OLTP workload. The mixed workload consists of a class of TPC-H and a class of TPC-C queries submitted by interactive clients, each class having a performance goal. Each client submits queries one after another with zero think time. The TPC-H database consists of 500MB of data. Four very large queries (queries 16, 19, 20 and 21) are excluded from the workload. The TPC-C database contains 50 warehouses. Workload intensity is controlled by the number of clients for each class (see Fig. 9). Each test ran for 12 hours and consists of 9 80-minute periods.

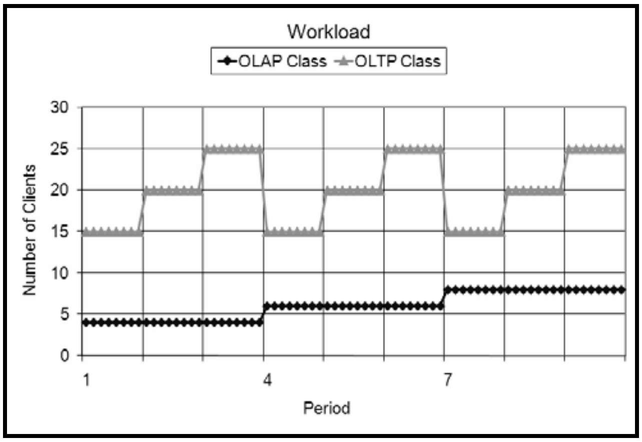

Fig. 9. Workload

The OLAP class is assigned with a lower importance level, for example 1. The OLTP class is assigned a higher importance level, for example 2. The heaviest workload is in period 9 where 8 clients from the OLAP Class and 25 clients from the OLTP class are issuing queries simultaneously.

5.2 Experimental Results

The set of experiments show the proposed approach in dealing with mixed workload relative to that of DB2 QP. The analysis of the results is discussed in Section 5.3. In all experiments, we use the workload shown in Fig. 9.

No Class Control. In this experiment, no control is exerted over the workload except for the total cost limit. This experiment serves as our baseline measure to observe how the performance changes with the changes of workload. The result is shown in Fig. 10.

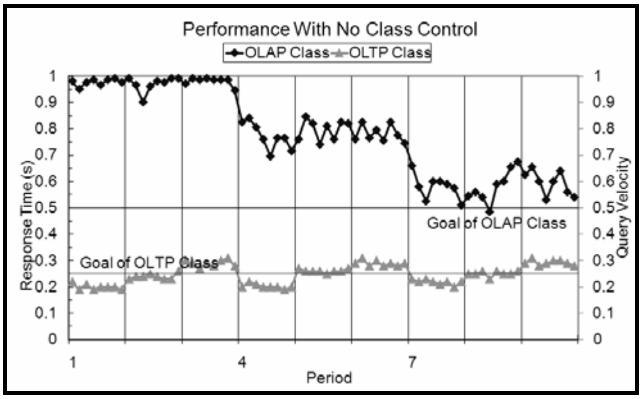

Fig. 10. Performance with No Class Control

Class Control with DB2 QP. In this experiment, we use DB2 QP as the performance controller. Because the DB2 QP imposes significant overhead on the OLTP class, there is no control over the OLTP class. Using the typical query control strategy of DB2 QP, the OLAP queries are partitioned into three groups based on the cost of the queries: large, medium and small. Queries whose cost is in the top 5% of the workload are placed in the large group; queries whose cost is in the next 15% are placed in the medium group and the remaining queries are placed in the small query group. The result is shown in Fig. 11.

Class Control with Query Scheduler. This experiment uses Query Scheduler to control performance. The performance goals for OLAP Class and OLTP Class are set as 0.5 (velocity) and 0.25 seconds (average response time) respectively. The total cost limit is 300000 timerons. Class control is performed by setting the cost limit for OLAP class, which is calculated during execution according to the performance of each workload class and predefined utility functions. In other words, the cost limit for OLAP class is calculated by optimizing the objective function. The results are shown in Fig. 12 for the performance and in Fig. 13 for the adjustment of the cost limit for both classes.

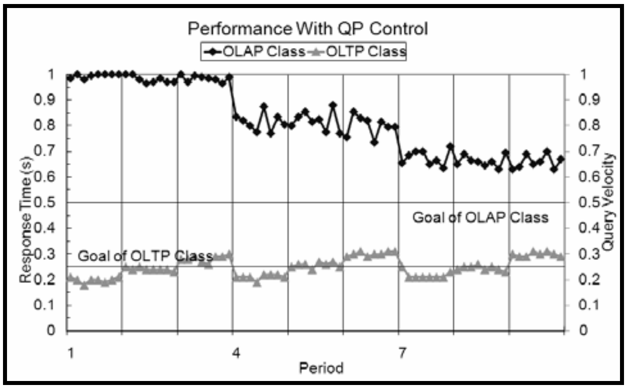

Fig. 11. Performance with DB2 QP Control

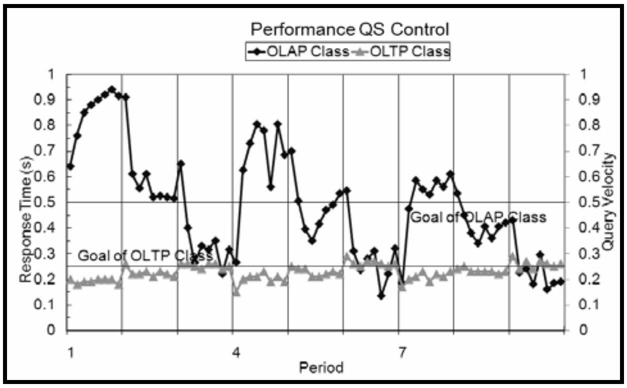

Fig. 12. Performance with Query Scheduler Control

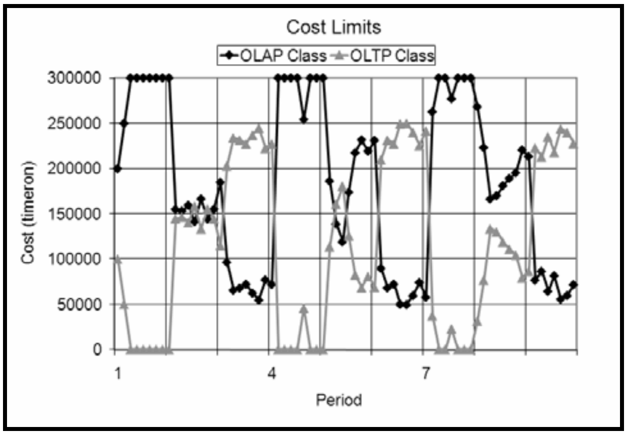

Fig. 13. Adjustment of Class Cost Limits with Query Scheduler Control

5.3 Analysis of the Experimental Results

Scalability of the New Approach. Although DB2 QP can directly control OLTP class, the heavy overhead incurred by directly controlling OLTP class makes it unscalable. Both DB2 QP and Query Scheduler can control OLTP class indirectly through directly control OLAP class. This provides a scalable approach for controlling OLTP class. DB2 QP can only set a static cost limit for OLAP class, while Query Scheduler can adjust cost limits for both classes dynamically (Fig. 13) to reflect the workload changes.

Differentiated Services. Query Scheduler with the proposed scalable approach can deliver differentiated service and DB2 QP can only provide limited differentiated service, while no-class-control can not. As shown in Fig. 12 for Query Scheduler, the OLTP class can better meet its performance goals than the OLAP class because the OLTP class is more important than the OLAP class. Since DB2 QP cannot dynamically adjust the cost limit for the OLAP class, the differentiated service that DB2 QP can provide is to set a static cost limit for the OLAP class, ignoring any fluctuations of both workload classes. For periods 3, 6 and 9 in Fig. 11, the cost limit for the OLAP class is set too high, because the important class, the OLTP class cannot meet its performance goal.

Adaptability. Query Scheduler with the proposed scalable approach can adapt to the workload changes to provide better differentiate service than DB2 QP. DB2 QP sets a static cost limit on the OLAP class to control its resource consumption and reserve resources for the OLTP class. However DB2 QP cannot adjust the limit to reflect resource requirements of the importance class - the OLTP class. As shown in Fig. 11, the performance goal of the OLTP class is always missed during the periods 3, 6 and 9 when the intensity of OLTP workload is high. Query Scheduler can detect workload changes for both classes and adjusts the cost limits for each workload class by maximizing the system utility. As shown in Fig. 12, the important class, the OLTP class, meets its performance goal almost all the time.

Dynamic Resource Allocation. From Fig. 13, we observe that Query Scheduler adjusts the class cost limits according to the workload changes. A higher class cost limit means more resources are allocated to the class. The amount of resources allocated to a class is based on its need to meet its performance goal and trade off with the other class. In the case of DB2 QP control, both classes are given a fixed amount of resources.

To conclude, the proposed approach can avoid the high overhead related to controlling the OLTP class and is effective for adapting mixed workload. It is able to respond to the workload changes using admission control to give preference to the important class.

6 Conclusions and Future Work

In this paper we briefly introduce AWMF, a general framework for workload adaptation, and its prototype implementation – Query Scheduler. We then investigate the overhead incurred by controlling workloads and propose a scalable approach for adapting mixed workload for Query Scheduler. The proposed approach indirectly controls OLTP workload by directly control OLAP workload. A general form of utility functions is proposed to reduce the pain to find the proper utility functions. Through a set of experiments we have shown that the proposed approach can avoid the high overhead related to controlling the OLTP class and is effective for adapting mixed workload.

In the future, we plan to study the possibility to implement the control mechanism inside the DBMSs, since this is the most effective way to control workload. We need to further investigate the overhead incurred by directly controlling workload and the approaches to avoid the overhead. Performance modeling for cost-based resource allocation is another issue need to be addressed. The accuracy of the performance models is the centre concern.

Acknowledgments. This research is supported by the IBM Centre for Advanced Studies (CAS), Toronto, the Natural Sciences and Engineering Re-search Council of Canada, the Centre for Communication and Information Technology, a division of Ontario Centres of Excel-lence Inc., and Shanxi Scholarship Council of China.

References

1. D.H. Brown Associate Inc.: HP Raises the Bar for UNIX Workload Management, http://whitepapers.silicon.com/0,39024759,60104905p-39000654q,00.htm
2. IBM Corporation: MVS Planning: Workload Management, 7th edn. (2003)
3. IBM Corporation: DB2 Query Patroller Guide: Installation, Administration, and Usage (2003)
4. Lo, T., Douglas, M.: The Evolution of Workload Management in Data Processing Industry: A Survey. In: Proceedings of 1986 Fall Joint Computer Conference, Dallas, TX, USA, pp. 768–777 (1986)
5. Menascé, D.A., Almeida, V.A.F.: Capacity Planning for Web Performance: Metrics, Models, and Methods. Prentice Hall, Upper Saddle River (1998)
6. Niu, B., Martin, P., Powley, W.: Towards Autonomic Workload Management in DBMSs. Journal of Database Management 20(3), 1–17 (2009)
7. Pacifici, G., Spreitzer, M., Tantawi, A., Youssef, A.: Performance Management for Cluster Based Web Services. IEEE Journal on Selected Areas in Communications 23(12), 2333–2343 (2005)
8. Schroeder, B., Harchol-Balter, M., Iyengar, A., Nahum, E.: Achieving Class-based QoS for Transactional Workloads. In: Proceedings of the 22nd International Conference on Data Engineering, p. 153 (2006)
9. Transaction Processing Performance Council, http://www.tpc.org