Adapting Mixed Workloads to Meet SLOs in Autonomic DBMSs

Baoning Niu, Patrick Martin, Wendy Powley
School of Computing, Queen's University
Kingston, Ontario, Canada, K7L 3N6
{niu | martin | wendy}@cs.queensu.ca

Paul Bird, Randy Horman
IBM Toronto Lab, 8200 Warden Ave,
Markham, Ontario, L6G 1C7
{pbird | horman}@ca.ibm.com

Abstract

Workload adaptation allows an autonomic database management system (DBMS) to efficiently make use of its resources and meet its Service Level Objectives (SLOs) by filtering or controlling the workload presented to it. Workload adaptation has been shown to be effective for OLAP and OLTP workloads. We outline a framework of workload adaptation and explain how it can be extended to manage mixed workloads comprised of both OLAP and OLTP queries. Experiments with IBM® DB2® Universal Database™ are presented that illustrate the effectiveness of our techniques.

1. Introduction

Autonomic Database Management Systems (DBMSs) face the problem of guaranteeing service level objectives (SLOs) for their complex workloads. The emerging trend of server consolidation results in a set of workloads with diverse and dynamic resource demands and competing performance objectives. Web-based applications introduce a need for flexible and guaranteed application service levels [1]. In order to meet the SLOs, Autonomic DBMSs must be able to recognize workload changes and allocate resources accordingly.

Workload adaptation is a technique that can be used to implement SLO compliance. It allows an autonomic DBMS to efficiently make use of its resources by filtering or controlling the workload presented to it. We previously proposed a general framework for workload adaptation in autonomic DBMSs and showed that it can effectively control OLAP workloads [4]. A similar framework by Schroeder et al controls OLTP workloads based on multiprogramming levels (MPL) by intercepting queries and performing admission control [5].

Mixed workloads consisting of both OLAP and OLTP queries are common. Controlling mixed workloads is more complex than controlling homogenous workloads. Control of OLAP workloads based on costs, the resource demand estimated by query optimizer, is appropriate because the requirements of OLAP queries vary 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 significant for OLTP queries with sub-second execution time and could be significantly larger than the execution time.

This paper explores techniques that can be used under the general framework of workload adaptation for a mixed workload using cost-based workload control. Section 2 briefly describes the general framework and its prototype implementation, Query Scheduler, for workload adaptation in autonomic DBMSs. Section 2 also formulates the problem to be solved. Section 3 discusses how OLTP workloads are handled in the framework and introduces some techniques for solving the problem. The evaluation of Query Scheduler for a mixed workload is outlined in Section 4. We conclude and suggest future work in Section 5.

2. Workload adaptation framework

We view workload adaptation in general as consisting of two processes, workload detection and workload control. Workload detection identifies workload changes by monitoring and characterizing current workloads and predicting future workload trends. Workload control involves effective allocation of resources using workload scheduling. 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, we first determine whether the resource allocation needs to be adjusted. If this is the case, then we use a performance model to evaluate possible workload scheduling plans and choose the optimal plan to exert control over the workload.

The Query Scheduler shown in Figure 1 is a prototype implementation of the workload adaptation framework with IBM® DB2® Universal Database system Version 8.2 (DB2 UDB). It uses Query Patroller [3] (DB2 QP), the workload controller in DB2 UDB, to intercept queries, acquire query information, and to release queries. DB2 QP is configured to automatically intercept all queries, record detailed query information, and block the DB2 agent responsible for executing the query until an explicit operator command is received.

The Monitor collects the information about the query from the DB2 QP control tables, including the query identification, query cost and query execution information. The Monitor passes the query information to the Classifier and to 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.

Workload control in the Query Scheduler is cost based. A scheduling plan is therefore expressed as a set of class cost limits, which determine the number of queries of each class that can execute at any one time. Cost limits are expressed in timerons, which is a generic cost measure used by the DB2 UDB optimizer to express the combined resource usage to execute a query. The cost limit for a service class represents the maximum allowable total cost of all concurrent executing queries belonging to a service class. The sum of all class cost limits must not exceed the system cost limit, which in our case is determined experimentally by plotting the curve of the throughput versus the system cost limit to ensure the system running in a healthy state or under-saturated [4].

The Dispatcher follows a scheduling plan by releasing queries for execution as long as the addition of a new query does not mean that the cost limit for the

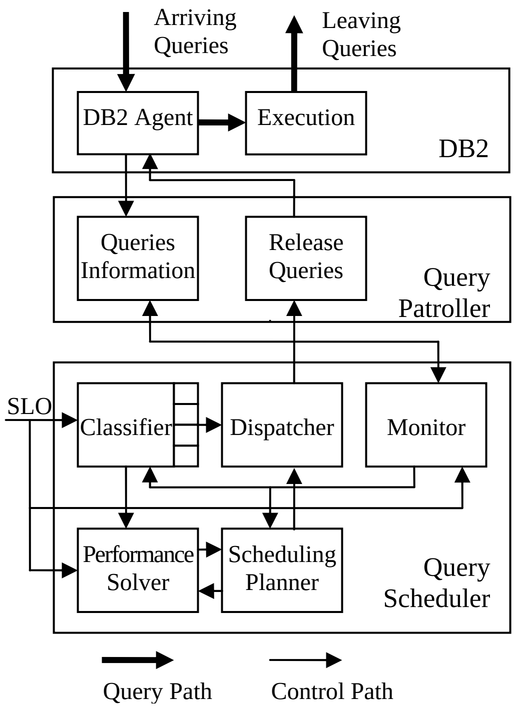

Figure 1. Query Scheduler

query's class is exceeded. The Dispatcher releases a query for execution by calling the unblocking API provided by DB2 QP, which releases the blocked agent. The Scheduling Planner consults with the Performance Solver at regular intervals to determine an optimal scheduling plan, and passes this plan to the Dispatcher. We use utility functions to capture the goals and importance of a workload and then view the development of a scheduling plan as an optimization problem involving the utility functions [4].

3. Adapting OLTP workloads

We found that using Query Scheduler to directly adapt an OLTP workload was not practical because of the overhead involved in intercepting and managing the queries. This overhead significantly outweighed the sub-second execution time of the OLTP queries and unfairly biased the results of any experiment.

With a mixed workload, however, 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 limits of competing OLAP classes and can decrease resources allocated to

OLTP queries by raising the cost limits of competing OLAP classes.

3.1. Performance metrics

The most widely used performance metrics for DBMS workloads 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 as compared to its total time in the system, is a better choice.

In this paper, we use average response time as the performance and goal metric for OLTP workloads and query velocity as the performance and goal metric for OLAP workloads. We define *query velocity* as

$$Query\_Velocity = Execution\_Time / Response\_Time$$

where *Execution_Time* is the amount of time a query is running in the DBMS and *Response_Time* is the time from when a client issues a query until it receives a response. *Response_Time* therefore includes *Execution_Time* plus the time the query is held up by the workload adaptation mechanism.

Query velocity is a ratio with a value between 0 and 1. It is a measure of how fast work is running compared to ideal conditions without delays, and indicates the impact of the workload adaptation mechanisms. A larger value means a shorter waiting time compared with execution time and hence better performance.

3.2. Performance modeling

The performance model for OLAP workload classes is:

$$V_i^k = \begin{cases} V_i^{k-1} C_i^k / C_i^{k-1} & \text{if } V_i^{k-1} C_i^k / C_i^{k-1} \le 1 \\ 1 & \text{if } V_i^{k-1} C_i^k / C_i^{k-1} > 1 \end{cases}$$

where V_i^{k-1} and V_i^k are the query velocity of service class i at the $(k-1)^{st}$ and k^{th} control intervals, respectively. C_i^{k-1} and C_i^k are the class cost limits of service class i at the $(k-1)^{st}$ and k^{th} control intervals, respectively [4].

We cannot use the same model for OLTP workloads for several reasons. First, the performance metrics are different. OLAP workload classes use query velocity, while the OLTP workload class uses average response time. Second, the system does not control the OLTP class directly so the total cost of OLTP queries at any one time is unknown. Third, OLAP queries tend to be I/O intensive whereas OLTP queries are CPU intensive.

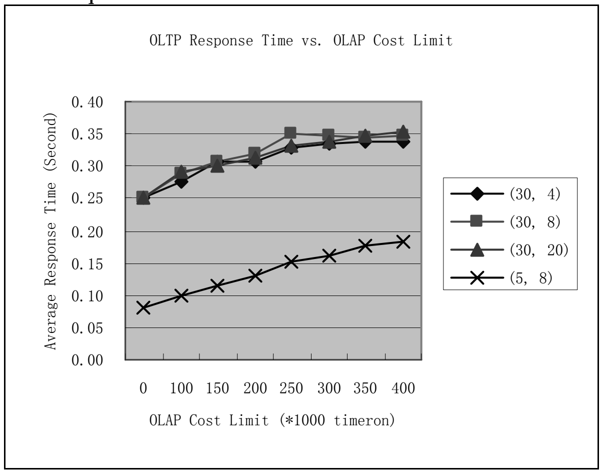

Figure 2. OLTP performance vs. OLAP cost limit

We justified our decision to indirectly manage OLTP workloads by directly managing the OLAP workloads experimentally. We measured the average response time of the OLTP class relative to the sum of cost limits of all OLAP classes (Figure 2). The number pairs shown in the legend indicate the number of OLTP clients, and the number of OLAP clients. The average response time of the OLTP class is almost linear with the increase of the total cost limit of OLAP classes when the system is not overloaded (system total cost less than 300K timerons).

Based on this knowledge, we can use the following linear equation to model the performance of the OLTP workload:

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

where t^{k-1} and t^k are the average response times of the OLTP class at the $(k-1)^{st}$ and k^{th} control intervals, respectively, C^{k-1} and C^k are the class cost limits of the OLTP class at the $(k-1)^{st}$ and k^{th} control intervals, respectively, and s is a constant that is obtained using linear regression.

3.3. Monitoring the OLTP workload

It is necessary to monitor the performance, that is the average response time, of the OLTP workload in order to make control decisions for the mixed workload. Since we turned off DB2 QP for OLTP workload, we need other approaches for acquiring the information. The DB2 UDB snapshot monitor records the execution time of the most recently finished query for a client. We, therefore, can take snapshots at fixed intervals, for example every 10 seconds, to get samples of response times of OLTP queries from all the clients and average them to get the average response time of the OLTP workload. The sampling interval must not be too small, which will incur significant overhead, nor too large, which would decrease accuracy.

4. Experiments

In this section we describe a set of experiments to study the effectiveness of Query Scheduler in providing differentiated service to mixed workload classes.

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

The TPC-H and TPC-C [6] benchmarks served as the OLAP and OLTP workloads, respectively, for our experiments. The TPC-H database consists of 500MB of data. Four very large queries (queries 16, 19, 20 and 21) are excluded from the TPC-H workload. The TPC-C database contains 50 warehouses. The tables for the two workload types were placed in separate databases. This allows us to focus on the impact of the workload adaptation on the allocation of system resources, such as CPU and I/O, while ignoring other sources of contention between OLTP and OLAP workloads, such as buffer pools and lock lists.

The mixed workload consisted of three workload classes; two classes of TPC-H and one class of TPC-C queries submitted by interactive clients. Each class has a performance goal. Each client submitted queries one after another with zero think time. Workload intensity was controlled by the number of clients for each class

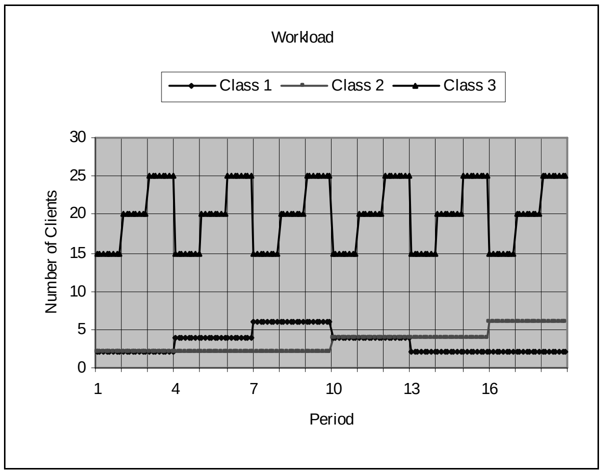

Figure 3. Workload

(Figure 3). The number of clients for an OLAP class varied from 2 to 6 at any one time and the number of clients for the OLTP class varied from 15 to 25 at any one time. Each test run lasted 24 hours and was broken down into 18 80-minute periods. Workload intensity was consistent within a given period.

Class 1 and Class 2 were OLAP classes with importance levels of 1 and 2, and query velocity goals of 0.4 and 0.6, respectively. Class 2 is more important than Class 1 and therefore had a higher query velocity goal, which means that its queries should be delayed less than queries of Class 1. Class 3 was the OLTP class with the highest importance level of 3, and was assigned average response time goal 0.25 seconds as its performance goal. We determined experimentally that the goals for the three classes were reasonable. We see from Figure 3 that the heaviest workload was in period 18 where two clients from Class 1, six clients from Class 2 and twenty-five clients from Class 3 were issuing queries simultaneously.

4.1. Results

The following set of experiments show the relative effectiveness of Query Scheduler, with dynamic workload adaptation, over the static control of DB2 QP to handle mixed workloads. In all experiments, we use the workload shown in Figure 3.

4.1.1. No class control. In this experiment, no control was exerted over the workload except for the system cost limit. This experiment serves as our baseline measure to observe how the performance changes with the variations in the workloads. The results are shown in Figure 4.

4.1.2. Class Control with DB2 QP. In this experiment, we use DB2 QP as the performance controller. DB2 QP imposes significant overhead on sub-second queries in the OLTP class so it is turned off for Class 3. Using the typical query control strategy of DB2 QP, the OLAP queries are partitioned into three groups (large, medium and small) based on the cost of the queries. 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.

In order to observe how DB2 QP provides differentiated services, we ran experiments with priority control turned on and off. For the case where priority control was turned on, we set the priority of Class 2 higher than that of Class 1. The results of this experiment are shown in Figure 5. For the case where priority control was turned off, we observed that the performance was similar to the case with no control so the results are not presented here.

4.1.3. Class control with Query Scheduler. This experiment uses Query Scheduler to control performance. The total cost limit is 300,000 timerons. Class control is performed by setting class cost limits. The sum of all class cost limits is equal to the total system cost limit. Class cost limits are calculated during execution according to the performance of each workload class and predefined utility functions. In other words, class cost limits are calculated by optimizing the objective function. The performance results are shown in Figure 6. Figure 7 shows the adjustment of class cost limits.

4.2. Analysis

Control over OLTP workload: High overhead makes it impractical to directly control the OLTP workload using the current test framework. Both DB2 QP and Query Scheduler can control the OLTP workload indirectly by directly controlling the OLAP workload. DB2 QP can only set a static cost limit for the OLAP workload, while Query Scheduler can adjust cost limits for all service classes dynamically (Figure 7) to reflect the workload changes.

Differentiated services: DB2 QP can only provide limited differentiated service within the OLAP classes by assigning priorities. As shown in Figure 5, Class 2 always performs better than Class 1. Figure 6 shows that Query Scheduler can provide differentiated service to both OLAP and OLTP classes. Class 3 meets its

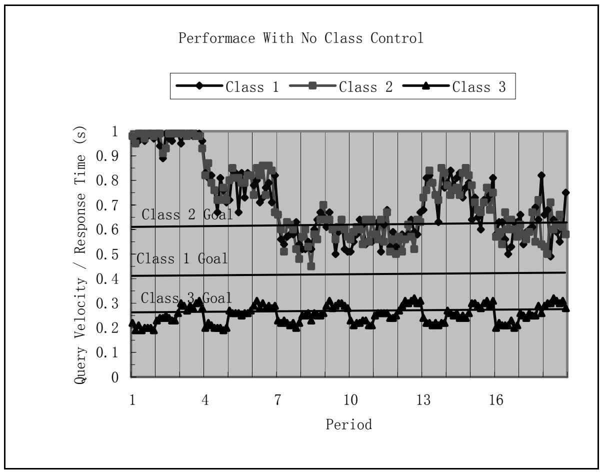

Figure 4. No class control

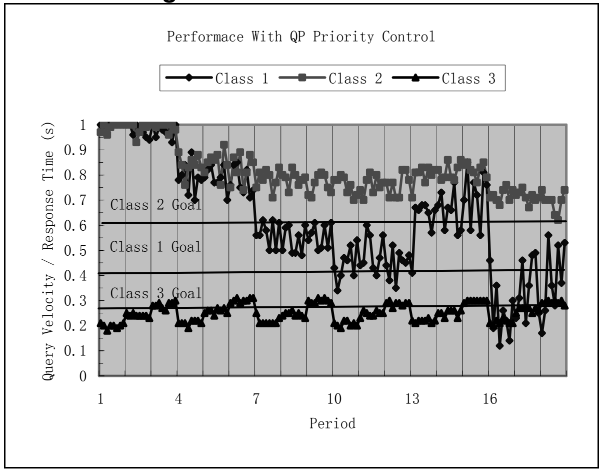

Figure 5. DB2 QP priority control

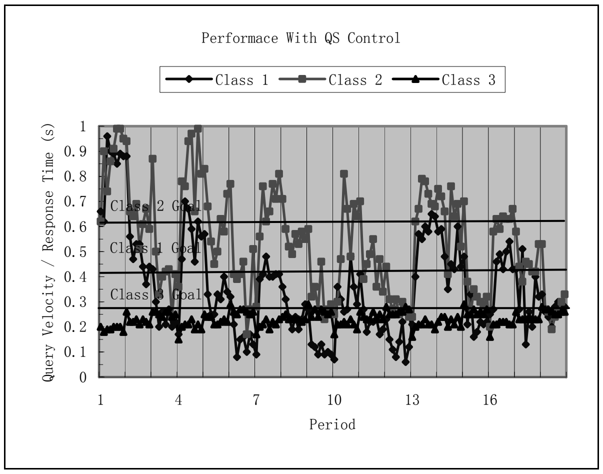

Figure 6. Query Scheduler control

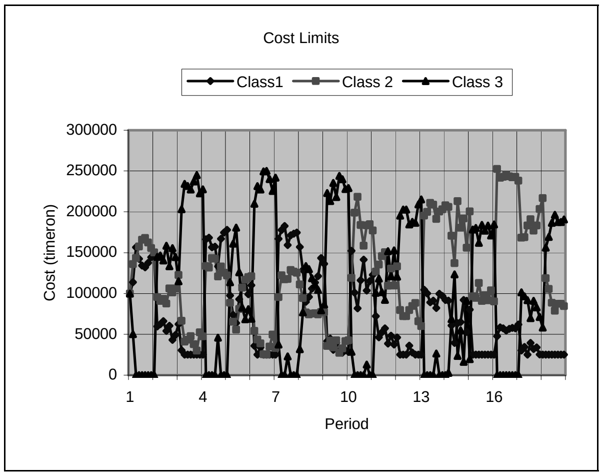

Figure 7. Adjustment of class cost limits with Query Scheduler control

performance goal nearly all the time. The performance of Class 2 is better than that of Class 1 in most cases.

Quality of differentiated service: Query Scheduler can provide better differentiated service than DB2 QP. DB2 QP sets a static cost limit on the OLAP workload to to control its resource consumption and gives higher priority to Class 2 over Class 1. DB2 QP cannot adjust the limit to reflect resource requirements of Class 3. As shown in Figure 5, the performance goal of Class 3 is always missed during periods 3, 6, 9, 12, 15 and 18 where the intensity of OLTP workload is high, and during period 17 where the intensity of OLTP workload is medium and the intensity of OLAP workload is high.

Query Scheduler can detect workload changes for both the OLAP and OLTP workloads and adjust cost limits for each service class by maximizing the system utility. As shown in Figure 6, Class 3 meets its performance goal when its workload intensity is medium and low and oscillates around its performance goal when its workload intensity is high. We also notice that the performance of Class 2 is better than that of Class 1 in most cases. This means Query Scheduler can provide better differentiated service for a mixed workload than DB2 QP.

Importance of classes: Query Scheduler supports the concept of business importance. A high importance level does not mean that the service class always possesses more resources than those with low importance levels. The importance level of a class is in effect only when the class violates its performance goals and is not synonymous with priority. As shown in Figure 7, Class 3, which has the highest importance level, possesses few resources when its workload intensity is low (periods 1, 4, 7, 10, 13 and 16) and it can meet its performance goal. During periods 3, 6, 9, 12, 15 and 18, its workload intensity is high and its performance goal is violated. In response, Query Scheduler assigns more than half of the total resources to Class 3 to meet its performance goal. In period 18, the workload intensity is the heaviest, however, the cost limit of Class 3 is less than that in periods 3, 6 and 9 where the workload intensity of Class 3 is same as in the period 18. This is because the workload intensity of other classes is the heaviest in period 18 and the competition for resources is more severe than the other periods.

Dynamic resource allocation: Using DB2 QP with priority control, Class 2 always has the privilege to possess more resources, even when it exceeds its performance goal, as shown in Figure 5. Query Scheduler, however, as shown in Figure 7, adjusts the class cost limits according to the workload changes thus allocating resources to where they are needed at the appropriate time. 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 among other classes.

To conclude, our framework for workload adaptation in autonomic DBMSs is effective for mixed workloads. It is able to respond to the workload changes using admission control to give preference to important service classes, or to the service classes whose performance goals are violated.

5. Future Work and Conclusions

In this paper we discussed the techniques for workload adaptation with a mixed workload. We briefly introduced the general framework and its prototype implementation, Query Scheduler, for workload adaptation in autonomic DBMSs. Our primary focus involved enforcing indirect control over OLTP workload by directly controlling the OLAP workload. Through a set of experiments we have shown the effectiveness of the framework for mixed workload.

In the future, we plan to address the issue of the overhead of controlling OLTP workloads. The most effective way to manage performance of OLTP workload is to directly control it. One approach is to implement the control mechanism inside the DBMS itself. Performance modeling for OLTP workload is another issue that needs to be addressed when direct control over OLTP workload is feasible. Cost-based resource allocation is somehow inaccurate. Estimating the resource demands of a query is the ultimate solution.

Acknowledgements

This research is supported by the IBM Centre for Advanced Studies (CAS), Toronto, the Natural Sciences and Engineering Research Council of Canada, and the Centre for Communication and Information Technology, a division of Ontario Centers of Excellence Inc.

Trademarks

References

[1] D. H. Brown Associate, Inc., "HP Raises the Bar for UNIX Workload Management", 2004, http://whitepapers.silicon.com/0,39024759,60104905p39000654q,00.htm.

[2] IBM Corporation, *MVS Planning: Workload Management*, 7th edition, Oct. 2003.

[3] IBM Corporation, *DB2 Query Patroller Guide: Installation, Administration, and Usage*, 2003.

[4] B. Niu, P. Martin, W. Powley, R. Horman, and P. Bird, "Workload Adaptation in Autonomic DBMs", *Proceedings of CASCON 2006*, Toronto, Canada, Oct. 16 – 19, 2006, pp 161 - 173.

[5] B. Schroeder, M. Harchol-Balter, A. Iyengar, and E. Nahum, "Achieving Class-based QoS for Transactional Workloads", *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06),* Apr. 03 - 07, 2006, pp 153.

[6] Transaction Processing Performance Council. http://www.tpc.org.