Workload Adaptation in Autonomic DBMSs

Baoning Niu, Patrick Martin, Wendy Powley

School of Computing, Queen's University, {niu | martin | wendy}@cs.queensu.ca

Randy Horman, Paul Bird

IBM Toronto Lab, { horman | pbird }@ca.ibm.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). This paper presents a framework and a prototype implementation of a query scheduler that performs workload adaptation in a DBMS. The system manages multiple classes of queries to meet their performance goals by allocating DBMS resources through admission control in the presence of workload fluctuation. The resource allocation plan is derived by maximizing the objective function that encapsulates the performance goals of all classes and their importance to the business. A first-principle performance model is used to predict the performance under the new resource allocation plan. Experiments with IBM® DB2® Universal Database™ are conducted to show the effectiveness of the framework.

1 Introduction

Performance management for database management systems (DBMSs) is becoming increasingly important to businesses as their workloads become more diverse and complex. The emerging trend of server consolidation has led to an environment with increased competition for shared resources between applications from potentially disjoint organizations in a single instance of the DBMS; this results in a workload with diverse and dynamic resource demands with competing performance objectives for these applications. In addition, Web-based applications introduce a need for flexible and guaranteed application service levels because they tend to involve unpredictable workloads, with a high rate of overall growth in workload size [6]. Allocating DBMS resources to competing workloads to meet performance objectives is a challenge. Simply maximizing overall resource utilization does not guarantee that individual performance objectives will be met. Performance management is complicated by the fact that the performance objectives for each application or workload class often have no strong relation to their resource demands, and vary widely. Workload classes with similar performance objectives might have different resource demands, while workload classes with similar resource demands may have different performance objectives. The business importance of each class must also be considered in performance management.

Most contemporary DBMSs have their own performance management component, for example, the IBM® DB2® Universal Database system (DB2 UDB) Query Patroller [8]. These controllers enforce efficient resource utilization through the use of statically determined priorities and thresholds on factors such as cost limits and MPLs (multiprogramming levels). The controllers do not automatically respond to workload changes and adjustments of performance goals.

Workload adaptation is 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 Service Level Objectives (SLOs). For example, if a DBMS is experiencing a heavy load from a less important application, it could delay these queries in order to allow queries from more important applications to meet their performance goals. Workload adaptation is a simple and effective method for workload control. It does not directly deal with resource allocation, and therefore, it does not require low-level resource control infrastructure in the DBMS or operating system support. Although workload adaptation does not directly manipulate resource allocations, it can still perform workload control at a fine granularity by admitting individual work requests.

Our work makes two main contributions to providing workload control in autonomic DBMSs. The first contribution is a general framework for performance-oriented workload adaptation in autonomic DBMSs. The framework classifies queries based on their performance goals and schedules the execution of queries from these classes based on the performance goals, the real performance, and resource utilization. The framework is based on a feedback loop that continually monitors system performance and the utilization of the various resources of the database system while, at regular intervals, determining the best scheduling plan that efficiently uses available resources to meet the different SLOs for the current workload.

The second contribution is a prototype implementation, called Query Scheduler, which adapts the workload for an instance of DB2 UDB. Query Scheduler manages multiple classes of queries to meet their SLOs by allocating DBMS resources through admission control in the presence of workload fluctuation. The resource allocation plan is derived by maximizing the objective function, which encapsulates the performance goals of all classes and their importance to the business. A first-principle performance model is used to predict the performance of the DBMS under the new resource allocation plan. We present experiments that evaluate the effectiveness of the Query Scheduler and compare it to the performance of an existing workload controller, namely DB2 Query Patroller (DB2 QP).

The rest of the paper is structured as follows. Section 2 describes related work. Section 3 explains our framework for workload adaptation and Section 4 discusses Query Scheduler, which is a prototype implementation of the framework. The evaluation of Query Scheduler is outlined in Section 5. We conclude and suggest future work in Section 6.

2 Related Work

The area of workload adaptation in DBMSs has been examined by a number of researchers. Brown et al. [3] propose an algorithm that automatically adjusts MPLs and memory allocation to achieve a set of per-class response time goals for a complex workload in DBMSs. The interdependency between classes that results from the competition for shared resources is solved by performance feedback. Pang et al. [14] propose an algorithm called Priority Adaptation Query Resource Scheduling to minimize the number of missed deadlines for a multi-class query workload, while at the same time ensuring that any deadline misses are scattered across the different classes according to an administratively-defined miss distribution. This objective is achieved by admission control, allocating memory and assigning priorities based on current resource usage, workload characteristics and performance experienced. Both of these approaches use heuristics to determine new workload control plans. Performance objectives are dealt with individually. In our study, we use performance objective encapsulation techniques to combine individual performance objectives into an objective function. It is optimized based on a performance model to find a solution for workload control.

Commercial systems currently support resource-oriented workload control. Teradata's Active System Management [2, 4] controls the workload presented to a DBMS by using predefined rules based on thresholds of the workload such as MPLs and number of users. DB2 QP [8] uses estimated query costs and MPLs to perform admission control. It can dynamically control the admission of queries against DB2 UDB databases so that small queries and high-priority queries can be run promptly, and system resources are used efficiently.

DB2 QP provides three mechanisms to help control query flow, namely, cost-based query classification, submitter queue prioritization, and threshold management. A query class is defined by specifying a cost range and an MPL threshold. Queries are assigned to query class based on the cost of query, which is the resource demand estimated by the query optimizer. The MPL threshold is the maximum number of queries in that class that can execute concurrently. When the threshold is reached, new queries are placed on the query class queue and are submitted for execution when the MPL falls below the threshold. This allows queries with different resource demands to be treated differently by specifying several query classes each with potentially different MPL thresholds thus using system resources more effectively. Submitter queue prioritization assigns high priorities to queries submitted by certain users so that these queries are run with shorter delays than others in the same query class queue. When the MPL threshold for a query class is reached, new queries are inserted into different positions in the query class queue based on the defined queue priority of the query submitter. This ensures that higher priority queries in a query class get submitted for execution first. By setting optional system level cost thresholds, DB2 QP automatically puts large queries on hold so that they can be cancelled or scheduled to run during off-peak hours. Unlike our approach, DB2 QP does not use performance objectives as guides.

Workload adaptation techniques have also been applied in the area of web services. Menascé et al. [11, 12, 13] propose a Quality of Service (QoS) Controller to manage workloads in an E-commerce environment. The QoS Controller adjusts system configuration parameters so that the Quality of Service requirements of the system are constantly met. The QoS Controller uses analytic performance models combined with combinatorial search techniques that run periodically to determine the best possible configuration for the system given its workload.

Pacifici et al. [15] present an architecture and prototype implementation of a performance management system for cluster-based web services. In this approach, web service workloads are partitioned into multiple service classes in each gateway and server resources are reactively allocated through admission control by adjusting MPLs for each gateway and service class to maximize the expected value of a given cluster utility function in the face of workload changes. As a function of the performance delivered to the various service classes, the cluster utility function plays a key role in providing differentiated service. Service levels are maintained by feedback control that incorporates a performance model.

Menascé and Pacifici both assume that the work requests are similar in size to simplify the performance model and perform admission control based on MPLs. Although this assumption may be valid in a web services environment, it does not hold true in DBMSs. Queries vary widely in size and in resource demand, which calls for more sophisticated performance models and admission control techniques.

3 The Framework for Workload Adaptation

Workload adaptation, as defined earlier, is a process of optimizing resource usage by controlling the workload presented to the system. As shown in Figure 1, we view workload adaptation to be composed of two processes, namely *workload detection* and *workload control*. The processes are in turn made up of four functional components - *workload characterization, performance modeling, workload control,* and *system monitoring*.

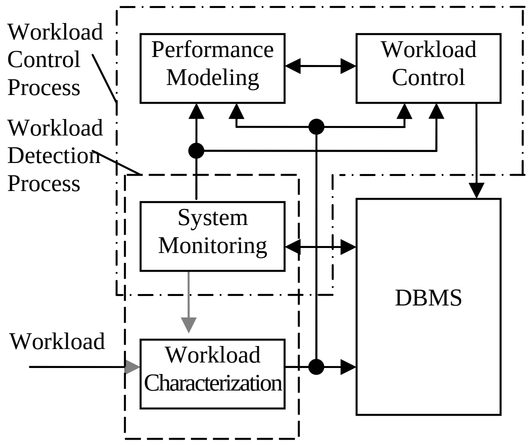

Figure 1: Framework for Workload Adaptation

3.1 Functional Components

Workload characterization is concerned with measuring and modeling production workloads [9, 10]. 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 [10]. The inputs to a performance model are workload parameters (such as resource demand and arrival rate) generated by the workload characterization component. The outputs are system performance and resource utilization parameters. In the autonomic era, systems are self-configurable, which calls for adaptive performance modeling techniques that evolve performance models in response to changes in the 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. Based on the support of the underlying system, the control plan can be a plan for direct resource allocation, parameter tuning or admission control. Control actions are triggered by workload changes.

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. The feedback information not only can be used as an indicator for workload changes to assist in workload characterization, but also aids in the evolution of performance models by tracking changes to systems [3, 7, 12].

3.2 Workload Detection Process

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

The workload characterization component partitions the workload, analyzes workload characteristics, and calculates resource demands with the help of feedback information from the system monitoring component. Partitioning the workload reduces the complexity of workload characterization by reducing the population to be probed. Analyzing workload characteristics, such as arrival rate and composition of workload components, helps to formulate a workload control strategy. Feedback information from system monitoring plays an important role in the process of workload detection. Alternatively, workload changes can be detected by monitoring the changes in performance and/or resource utilization [12, 15]. This is an effective approach to workload detection when workload characterization is impossible or too costly, or some characteristics cannot be directly derived from the workload itself.

3.3 Workload Control Process

Workload control involves system management via efficient allocation of resources. There are three approaches to workload control. First, direct resource allocation allocates a certain amount of resources to a workload, a workload class, or a single piece of work. Private memory for a process is usually allocated in this way. Second, parameter tuning regulates resources allocated to the work by changing the parameters related to resource usage. For example, increasing the buffer pool size in a DBMS improves the performance of an OLTP (On Line Transaction Processing) workload. There is no explicit assignment of resources to the OLTP workload, but an OLTP workload indeed benefits from increased buffer hit rates. Third, admission control regulates resource allocation by controlling the contention level on resources within a service class or across service classes. The more work requests that are admitted, the heavier the resource contention.

One of the main issues regarding workload control is how to determine the appropriate amount of control. This involves performance prediction under the suggested workload control plan or configuration. Performance administrators can determine the new configuration manually based on their experiences. Performance management systems require performance models to predict performance in order to be self-managing [5, 12]. 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 (Figure 1).

4 Query Scheduler

The Query Scheduler, which is shown in Figure 2, is a proof-of-concept implementation of the workload adaptation framework discussed above. It automatically controls the workload to DB2 UDB in order to satisfy defined performance goals.

Query Scheduler uses DB2 QP to intercept queries and acquire query information and, via direct commands to 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.

In our implementation 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. A query in a class queue is released only if the sum of the total cost of all executing queries of the service class plus the cost of the query does not exceed the class cost limit. The Dispatcher releases a query for execution by calling the unblocking API provided by DB2 QP, which releases the

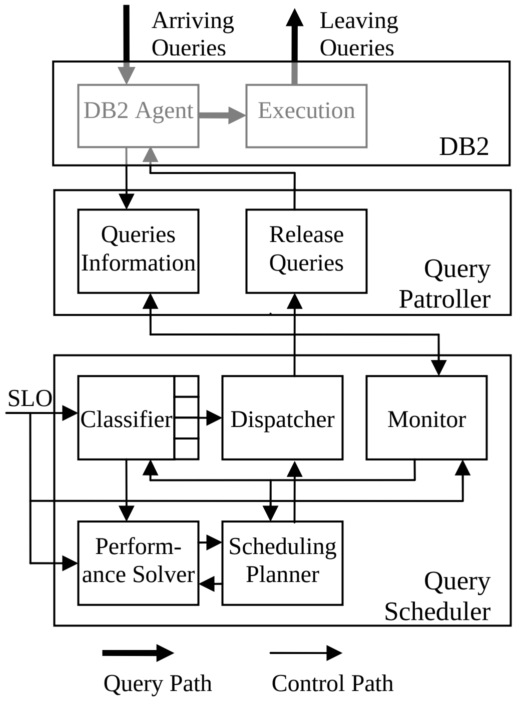

Figure 2: Query Scheduler

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.

4.1 SLO Encapsulation

Managing multiple performance goals with different business importance levels is complex. Not only does the complexity increase with the number of performance goals, but the interdependencies among the performance goals make it even more difficult to control the workload. It is desirable to collectively manage all the performance goals along with their business importance levels.

4.1.1 SLOs

SLOs are often specified by an importance level and a performance goal [1]. An importance level describes how important it is to the business to meet the performance goal relative to the other work competing for the same set of system resources. It identifies the order in which service classes should receive or donate resources when the system capacity is insufficient for all service classes to meet their goals. A performance goal defines the desired performance objective.

The most widely used performance goals are response time, throughput, and execution velocity [7]. Response time and throughput are well understood. These measures, however, are only useful when the work requests 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.

Because the Query Scheduler is currently implemented outside DB2 UDB, the overhead associated with managing queries with the Query Scheduler means that it is impractical to try to manage online transaction processing (OLTP) workloads, which are composed mainly of small queries. Instead, we focus on management of large queries, such as those found in decision support systems (DSSs). We therefore use the TPC-H DSS benchmark [16] as the workload in our experiments discussed below.

DSS workloads, such as TPC-H, contain queries with widely varying response times so a velocity-type goal is most appropriate. Motivated by the need to see a clear division between waiting and execution when performing admission control, we use the metric *Query_Velocity*, which we define as

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

The wait time for admission is dependent upon the policy governing admission control. If an admission control policy allows a query to be admitted earlier, the wait time for admission is small, otherwise, it is large. In order to make a meaningful comparison between different admission control policies, it is necessary to ensure that the expected execution time for a query is stable when the system is busy. We do this by setting a total cost limit for concurrently executing queries. Through experimentation, we found that a total cost limit of 300000 timerons is a reasonable saturation threshold (see Section 5.2) and we use this cost as the total cost limit in our experiments.

4.1.2 Objective Functions

Consider a system with *n* SLOs. Formally, an SLO is described as $\langle \bar{g}_i, i_i \rangle$, where $\bar{g}_i$ is the performance goal to be achieved and i_i is the importance level of the performance goal to be achieved. We denote $g_1, g_2, ..., g_n$ as the predicted performance given a workload control plan. The utility [15] of the i^{th} service class, u_i, describing how well the system meets an SLO, is the function of $\bar{g}_i, i_i$ and g_i : $u_i = f_i(\bar{g}_i, i_i, g_i)$. Multiple SLOs can be encapsulated into an objective function $f(u_1, u_2, ..., u_n)$. By properly choosing f_i and f, the workload control problem becomes one of optimizing the objective function f to find the optimal workload control plan.

In our implementation we choose the objective function as

$$f = \sum u_i \tag{1}$$

and the utility functions as

$$u_i = 1 - a_i^{\frac{\bar{g}_i - g_i}{g_i}} \tag{2}$$

where a_i is a constant indicating the importance of the service class. A larger a_i, denotes a higher degree of importance. From the shape of the utility function shown in Figure 3, we observe that as g_i increases, the curve becomes smoother. That is to say, when the service class achieves its

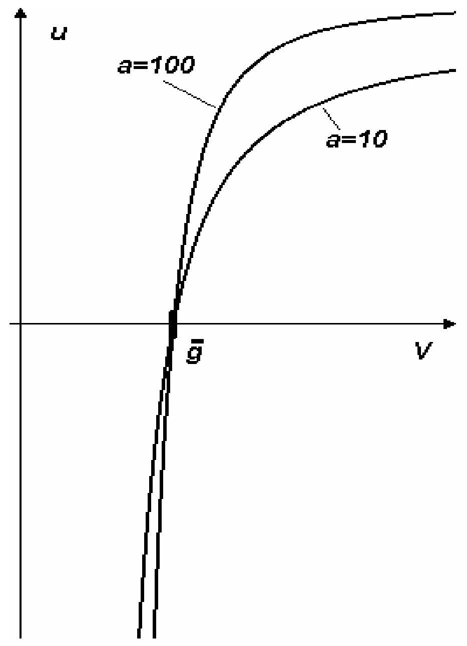

Figure 3: Utility Function

performance goal ($g_i \geq \overline{g}_i$), its utility increases slowly as its performance improves. This dictates that the system should not assign more resources to the service class when the service class is already meeting its performance goal. When g_i is decreasing, the curve becomes steeper. That is to say that when the service class violates its performance goal ($g_i < \overline{g}_i$), the marginal utility rapidly increases as more resources are assigned to the service class. In this case, allocating more resources to the service class should bring the class closer to meeting its performance goal.

4.2 Workload Characterization

The workload characterization component is the Classifier in our implementation. It identifies the characteristics, mainly resource demand, of a query, and classifies it into an appropriate service class. As shown in Figure 2, the information used in the Classifier includes SLOs, and query identification information and cost from the monitor component.

Resource demand must be acquired before making any control decisions in our framework. Acquiring resource demand for queries is difficult because a) queries vary widely in size, b) the resource demand of a query may be different during multiple executions because of the interference of other queries, and c) the resource demand of a query is dependent upon the configuration of the DBMS. Query costs estimated by the query optimizer roughly reflect the relative costs of queries in an ideal environment and we use these costs to represent the resource demand in the framework.

Query classification is based on performance goals. Each service class has a performance goal. The performance goals are predefined in SLOs and the query identification information from the monitor component is used to determine the performance goal of a query during runtime.

4.3 Performance Modeling

Admission control is based on the principle that system resources are shared by the queries running in the system. If the volume or multiprogramming level of service class A is larger than those of service class B, service class A is proportionally assigned more resources, and is given preference. The performance modeling problem is to predict the performance of a new admission control plan: a set of class cost limits.

Suppose there are m terminal users who submit queries one after another in a service class that is assigned a class cost limit of C^{k-1} at the $(k-1)^{th}$ control interval with the query velocity V^{k-1}. The performance modeling problem is to predict the query velocity V^k at the k^{th} control interval given the class cost limit C^k.

Suppose the average service time is T. As per queuing theory, the average queue length at the $(k-1)^{th}$ control interval is $m-1$. The average response time R^{k-1} and wait time W^{k-1} at the $(k-1)^{th}$ control interval are:

$$R^{k-1} = mT$$
$$W^{k-1} = (m-1)T$$

We have:

$$V^{k-1} = 1 - W^{k-1} \big/ R^{k-1} = 1/m$$

When the class cost limit becomes C^k, either the service time or the queue length is changed by a factor of $C^{k-1} \big/ C^k$ due to the change of the class cost limit. Now, the average response time and wait time at the k^{th} control interval is:

$$R^k = (C^{k-1} \big/ C^k)mT$$
$$W^k = ((C^{k-1} \big/ C^k)m - 1)T$$

We have:

$$\begin{aligned} V^k &= 1 - W^k \big/ R^k \\ &= 1 - ((C^{k-1} \big/ C^k)m - 1) \big/ ((C^{k-1} \big/ C^k)m) \\ &= C^k \big/ C^{k-1} * 1/m \\ &= V^{k-1} C^k \big/ C^{k-1} \end{aligned} \quad (3)$$

Now given the new class cost limit, we can predict the performance for the next control interval based on the performance and the class cost limit of the current control interval.

As shown in Figure 2, the performance modeling component is the Performance Solver in our implementation. It receives a new scheduling plan, a set of new class cost limits, from the Scheduling Planner and predicts the performance of each service class under this scheduling plan based on the performance of the current control interval from the Monitor component.

4.4 Workload Control

The workload control component finds an optimal scheduling plan and executes the plan. In our implementation, it consists of the Scheduling Planner and the Dispatcher. The Scheduling Planner finds an optimal scheduling plan at regular intervals and the Dispatcher executes it.

4.4.1 Performance Optimization

Finding an optimal scheduling plan can be described as following an optimization problem. We denote:

V_i : The performance goal of service class *i*

V_i^k : The performance of service class *i* at the k^{th} control interval

C_i^k : The class cost limit of service class *i* at the k^{th} control interval

u_i^k : The utility of service class *i* at the k^{th} control interval

C: The total cost limit allowed for all service classes

From equations (1), (2) and (3) we have:

$$f = \sum u_i^k \tag{4}$$

$$u_i^k = 1 - a_i^{\frac{V_i - V_i^k}{V_i^k}} \tag{5}$$

$$V_i^k = V_i^{k-1} C_i^k / C_i^{k-1} \tag{6}$$

Replacing V_i^k in (5) with (6) and u_i^k in (4) with (5), the objective function becomes the function of the new workload control plan:

$$f(C_1^k, C_2^k, ..., C_n^k)$$

with the constraint:

$$C_1^k + C_2^k + ... + C_n^k \le C .$$

If $f(C_1^k, C_2^k, ..., C_n^k)$ is a continuous function, we can use Lagrange method or searching techniques to solve it. Otherwise, we must solve it using searching techniques.

The Scheduling Planner receives SLOs and the query execution information from the monitor component, and predicts the performance of each service class by consulting the Performance Solver to find an optimal scheduling plan.

4.4.2 Admission Control

Admission control is performed by the Dispatcher when a new query arrives or when a query completes or aborts. The Dispatcher uses the following algorithm to execute the scheduling plan and perform admission control:

```
A query q arrived, completed or aborted;
$i \leftarrow$ the service class of q;
$C_i^T \leftarrow$ the total cost of concurrent queries
     of the service class $i$;
$C_i^L \leftarrow$ the class cost limit of service class $i$;
If (completed or aborted)
    $c \leftarrow$ the cost of q;
    $C_i^T \leftarrow C_i^T - c$;
$c \leftarrow$ the cost of the query at the front of
    the class queue $i$;
If ($C_i^T + c \le C_i^L$)
    Release the query at the front of the
    class queue $i$;
```

4.5 System Monitoring

The system monitoring component consists of the Monitor and a trigger in DB2 QP that informs the Query Scheduler of the arrival of new queries and the termination of running queries. When a query is submitted to the DBMS, the DB2 UDB agent responsible for the query informs DB2 QP that a new query has arrived. DB2 QP (with the threshold MAX_QUERY_ALLOWED set to 0) intercepts the query and blocks it. Whenever a query is intercepted, a new entry is added to the TRACK_QUERY_INFORMATION control table of DB2 QP to store the query information which includes query identification information, query execution information, the query cost, etc. The query execution information is updated whenever the query is completed or aborted, and is used for evaluating the performance of each service class.

A trigger on insertion or update defined on this table calls a stored procedure to connect to the Query Scheduler (the Monitor component) via a TCP socket to inform Query Scheduler that a new query was intercepted or completed. The Monitor watches the arrival and departure of queries, collects query identification data, performance data and resource usage data from DB2 QP and reports to the Classifier and the Scheduling Planner.

5 Experiments

In this section we describe a set of experiments to study the effectiveness of Query Scheduler in providing differentiated service to workload classes with different SLOs. We also compare it to the effectiveness of DB2 QP, which is typical of the level of control available in current DBMSs.

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. We use IBM DB2 UDB Version 8.2 and Query Patroller as supporting components.

5.1 Workload

As discussed above, we use the TPC-H standard DSS benchmark as our workload. The workload consists of two classes of TPC-H queries submitted by interactive clients or batch jobs, each class having a performance goal. Each client or batch job submits queries one after another with zero think time. The database consists of 500MB of data. Four very large queries (queries 16, 19, 20 and 21) are excluded from the workload. Workload intensity is controlled by the number of clients or batch jobs for each class (see Figure 4). Each test ran for 12 hours and consists of 6 2-hour periods.

Class 0 is deemed more important than Class 1. This is indicated by setting a stricter performance goal for Class 0 than for Class 1. The heaviest workload is in period 3 where 15 clients from Class 0 and 5 clients from Class 1 are issuing queries simultaneously.

5.2 The System Cost Threshold

The relationship between the total cost of concurrent queries in the system and the corresponding performance can be used to determine the system cost threshold – the total cost limit. Query admission is controlled by the total cost of active queries in the system and the corresponding average response time and throughput is calculated. The curves of total cost vs. average response time (Figure 5) and total cost vs. throughput (Figure 6) are plotted to determine the total cost limit that keeps the system saturated. We find that the circled point in Figure 5 and 6 with total cost limit of 300,000 timerons is

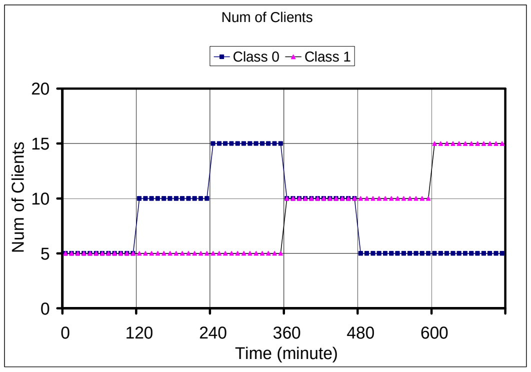

Figure 4: Workload

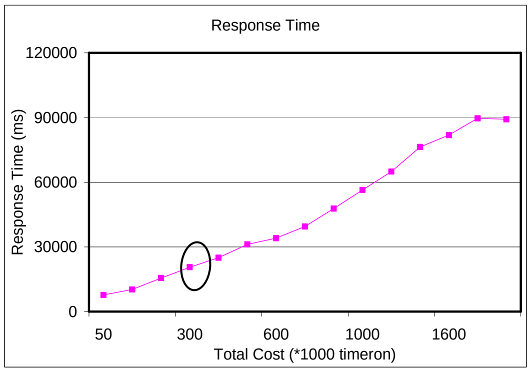

Figure 5: Response Time vs. Total Cost

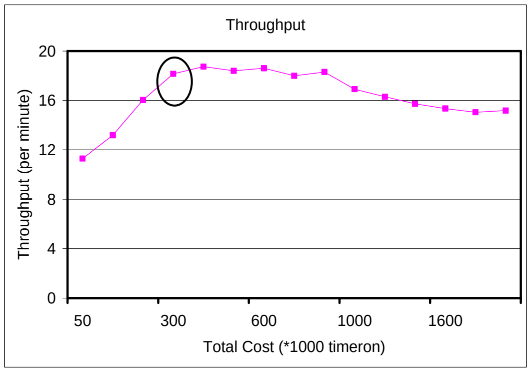

Figure 6: Throughput vs. Total Cost

the proper saturation point. If we increase the total cost limit further, we see a small increase in throughput, but we note that the average response time still increases linearly.

5.3 Experiments

The following set of experiments show the effectiveness of Query Scheduler relative to that of DB2 QP. The analysis of the results is discussed in Section 5.4. In all experiments, we use the workload shown in Figure 4.

5.3.1 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 Figure 7.

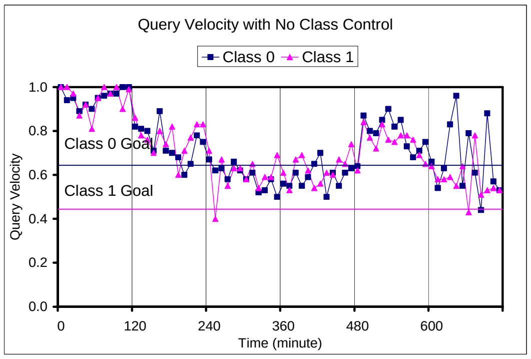

Figure 7: Query Velocity with No Control

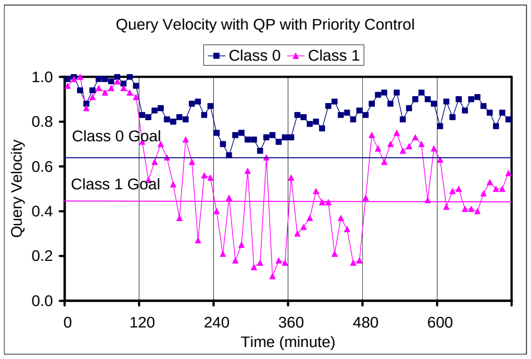

Figure 8: Query Velocity with DB2 QP with Priority Control

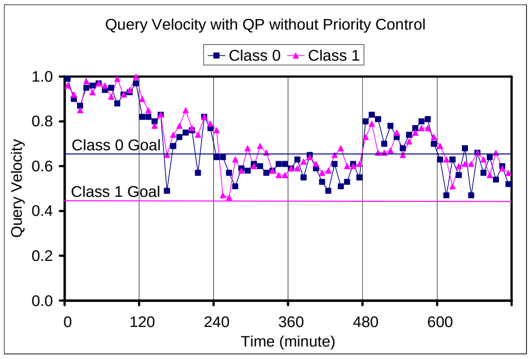

Figure 9: Query Velocity with DB2 QP without Priority Control

5.3.2 Class Control with DB2 QP

In this experiment, we use DB2 QP as the performance controller. The workload is partitioned into three groups based on the cost of queries: large, medium and small. The cost threshold for large group is chosen as the lowest percentile cost of 95 of all queries and 80 for medium group:

Large:

cost > the lowest percentile cost of 95

Medium:

the lowest percentile cost of 80 < cost ≤ the lowest percentile cost of 95

Small:

cost ≤ the lowest percentile cost of 80

In order to demonstrate how DB2 QP provides differentiated services, we first perform service class control by setting priorities for the two classes. The priority of Class 0 is higher than that of Class 1, for example 600 for Class 0, and 500 for Class 1.The result of this experiment is shown in Figure 8. We then turn off priority control. The result is shown in Figure 9.

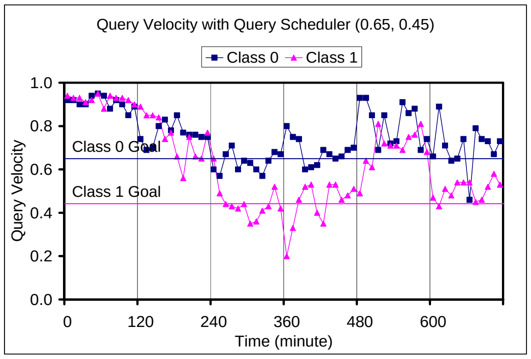

Figure 10: Query Velocity with Query Scheduler with Goals (0.65, 0.45)

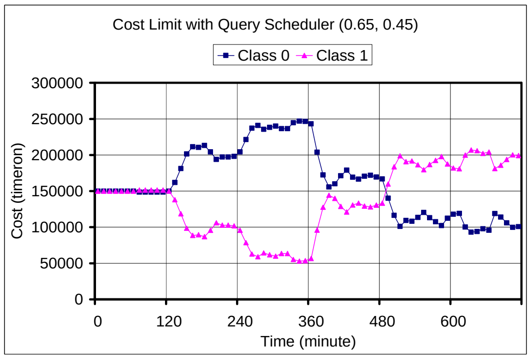

Figure 11: Adjustment of Class Cost Limit with Query Scheduler with Goals (0.65, 0.45)

5.3.3 Class Control with Query Scheduler

This experiment uses Query Scheduler to control performance. The performance goals for Class 0 and Class 1 are set as 0.65 and 0.45 respectively. The total cost limit is 300000 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 results are shown in Figure 10 for the query velocity and in Figure 11 for the adjustment of class cost limit.

To show the ability of Query Scheduler to adapt to the changes of performance goals, we ran a second experiment with a tighter performance goal (0.75) for Class 0. The results are shown in Figure 12 for the query velocity and in Figure 13 for the adjustment of class cost limits.

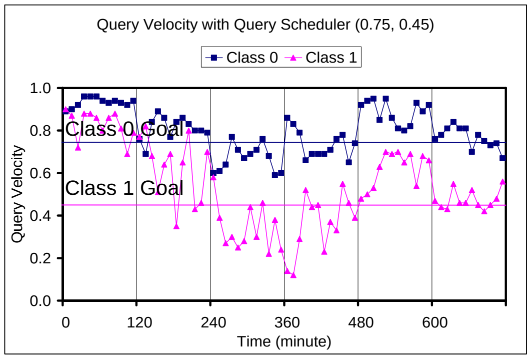

Figure 12: Query Velocity with Query Scheduler with Goals (0.75, 0.45)

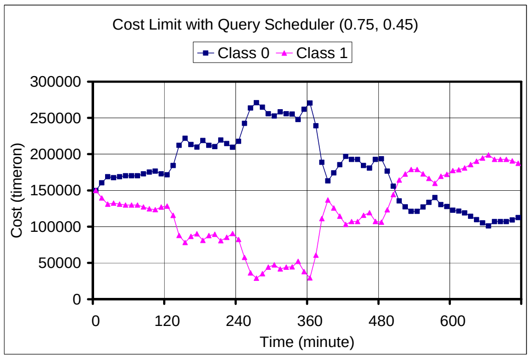

Figure 13: Adjustment of Class Cost Limit with Query Scheduler with Goals (0.75, 0.45)

5.4 Analysis of the Results

Differentiated services: The results of our experiments show that both DB2 QP and Query Scheduler can provide differentiated services, while No class control cannot. DB2 QP provides differentiated services by assigning different priorities to different service classes. As shown in Figure 8, with the higher priority assigned to Class 0, Class 0 always performs better than Class 1. When priority control is turned off as shown in Figure 9, the query velocity curves of both classes are similar to the case of No class control (Figure 7). As shown in Figure 10 and 12 for Query Scheduler, Class 0 can better meet its performance goals than Class 1 because Class 0 is more important than Class 1.

Quality of differentiated services: DB2 QP with priority control sets static priorities to different service classes (600 to Class 0, and 500 to Class 1). Class 0 is always given higher priority even when it is exceeding its performance goal and Class 1 is in violation of its goal in periods 2, 4 and 6 as shown in Figure 8. Query Scheduler dynamically adjusts the class cost limits based on the performance as shown in Figures 11 and 13. Although it always gives preference to the important class, Class 0, it never allocates too many resources to Class 0 to prevent Class 1 from meeting its performance goal if possible as shown in the periods 2, 4 and 6 in Figures 10 and 12. When the workload is too heavy to meet both performance goals in periods 3 and 4, DB2 QP with priority control cannot meet the performance goals for Class 1 as shown in Figure 8, while Query Scheduler is able to keep both classes converging on their performance goals as shown in Figure 10 and 12.

Importance of classes: We notice that Query Scheduler can assure that both classes converge on their performance goals when the performance goals are 0.65 and 0.45 (Figure 10). When the performance goal of Class 0 is changed to a tighter goal 0.75, Query Scheduler cannot meet the performance goals for both classes in periods 3 and 4 (Figure 12). However, Query Scheduler recognizes that Class 0 is more important than Class 1 and attempts to minimize the goal violations for the important class, to the detriment of Class 1, as seen in Figure 12. Although Class 0 is more important than Class 1, Query Scheduler can assign more resources to Class 1 than DB2 QP with priority control when

Class 0 meets its performance goals in periods 2 and 6. This means that the importance level of a class is in effect only when the class violates its performance goals and is not synonymous with priority.

Dynamic resource allocation: From Figures 11 and 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, as shown in periods 2, 5 and 6 in Figures 11 and 13. In the case of DB2 QP with priority control, Class 0 always has the privilege to possess more resources even when it exceeds its performance goal as shown in Figure 8.

To conclude, our framework for workload adaptation in autonomic DBMSs is effective. 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.

6 Future Work and Conclusions

In this paper we present a framework and prototype implementation – Query Scheduler, for workload adaptation in autonomic DBMSs. We use query cost as resource demand and perform admission control based on SLOs and system resource utilization. Class cost limits are determined dynamically through optimizing the objective function that encapsulates the SLOs with utility functions. Through a set of experiments we have shown the effectiveness of the framework.

In the future, we plan to repeat our experiments with different systems to check its universality. We plan to add an OLTP workload as well as additional service classes to examine the effectiveness of the framework with a more complex workload. The experiments we have conducted are based on a relative stable workload. We plan to experiment using a randomly changing workload and apply tracking techniques to track workload changes. Finally, we expect that using detailed costs (CPU cost and I/O cost) in place of total query cost will produce finer control over performance.

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 Centres of Excellence Inc.

About the Author

Baoning Niu is a PhD student from the School of Computing at Queen’s University. His research interests include: performance management for DBMSs, autonomic DBMSs.

Patrick Martin is a Professor and Associate Director of the School of Computing at Queen’s University. He holds a BSc from the University of Toronto, MSc from Queen’s University and a PhD from the University of Toronto. He is also a Faculty Fellow with IBM's Centre for Advanced Studies. His research interests include database system performance, Web services and autonomic computing systems.

Wendy Powley is a Research Associate and Adjunct Lecturer in the school of Computing at Queen’s University. She holds a BA in psychology, a BEd, and an MSc in Computer Science from Queen’s University. Her research interests include database systems, web services and autonomic computing.

Randy W. Horman, IBM Data Management Division, IBM Toronto Lab, 8200 Warden Ave, Markham, Ontario, L6G 1C7 (horman@ca.ibm.com) . Mr. Horman is a Senior Technical Staff Member on the DB2 development team at the IBM Toronto Lab. He received a B.A. degree in mathematics, computer science, and economics, as well as an M.Math degree in computer science from the University of Waterloo in 1994 and 1995, respectively. He subsequently joined IBM at the Toronto Lab, where he began working on the parallel database system, DB2 Parallel Edition. Recently, Mr. Horman has focused his attention on database manageability, and in particular the applicability of autonomic technology. Mr. Horman is a member of the Association for Computing Machinery and the Computer Society of the Institute of Electrical and Electronics Engineers.

Paul Bird is a Senior Technical Staff Member in the DB2 Universal Database for Linux®, UNIX®, and Windows® Development organization within the Information Management group of IBM. His areas of interest include workload management, security, and general SQL processing.

References

[1] G. Adam. "Understanding Workload Manager: Basic Metrics", 2001, http://www.naspa.com.

[2] C. Ballinger. "Introduction to Teradata's Priority Scheduler", 2002, http://www.teradatalibrary.com/pdf/eb3092.pdf.

[3] K. P. Brown, M. Mehta, M. J. Carey, and M. Livny. "Towards Automated Performance Tuning For Complex Workloads", Proceedings of the 20th Very Large Data Base Conference, Santiago, Chile, 1994, pp.

[4] D. P. Brown, A. Richards, R. Zeehandelaar, and D. Galeazzi. "Teradata Active System Management", http://www.teradata.com/t/page/145613/index.html.

[5] Y. Diao, F Eskesen, S. Froehlich, J. Hellerstein, A. Keller, L. Spainhower, and M. Surendra. "Generic On-line Discovery of Quantitative Models for Service Level Management", Proceedings of IFIP/IEEE 8th International Symposium on Integrated Network Management (IM 2003), Mar. 24 - 28, 2003, Colorado Springs, USA, pp. 157 - 170.

[6] D. H. Brown Associate, Inc. "HP Raises the Bar for UNIX Workload Management", 2004, http://whitepapers.silicon.com/0,39024759,60104905p-39000654q,00.htm.

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

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

[9] T. Lo, and M. Douglas. "The Evolution of Workload Management in Data Processing Industry: A Survey", Proceedings of 1986 Fall Joint Computer Conference, 1986, Dallas, TX, USA, pp. 768 - 777.

[10] D. A. Menascé, and V. A. F. Almeida. "Capacity Planning for Web Performance: Metrics, Models, and Methods", Prentice Hall, Upper Saddle River, NJ, 1998.

[11] D. A. Menascé, V. A. F. Almeida, R. Fonseca, and M. A. Mendes. "A Methodology for Workload Characterization of E-commerce Sites", Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Nov. 3-5, 1999, Denver, Colorado, USA, pp. 119 - 128.

[12] D. A. Menascé, and M. N. Bennani. "On the Use of Performance Models to Design Self-Managing Computer Systems", *Proceedings of 2003 Computer Measurement Group Conference*, Dec. 7-12, 2003, Dallas, TX. USA, pp. 1 - 9.

[13] D. A. Menascé, D. Barbará, and R. Dodge. "Preserving QoS of E-commerce Sites through Self-Tuning: A Performance Model Approach", *Proceedings of the 3rd ACM conference on Electronic Commerce*, Tampa, Florida, USA, Oct. 14 - 17, 2001, pp. 224 – 234.

[14] H. Pang, M. J. Carey, and M. Livny. "Multiclass Query Scheduling in Real-Time Database Systems", *IEEE Transaction on Knowledge and Data Engineering*, Vol. 7, No. 4, Aug. 1995.

[15] G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef. "Performance Management for Cluster Based Web Services", *IEEE Journal on Selected Areas in Communications*, Volume 23, Issue 12, Dec. 2005, pp 2333- 2343.

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

Trademarks