WORKLOAD ADAPTATION IN AUTONOMIC DATABASE MANAGEMENT SYSTEMS

by

Baoning Niu

A thesis submitted to the School of Computing

in conformity with the requirements for

the degree of Doctor of Philosophy

Queen's University

Kingston, Ontario, Canada

January, 2008

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). It is a challenge to adapt multiple workloads with complex resource requirements towards their performance goals while taking their business importance into account. This thesis studies approaches and techniques for workload adaptation.

First we build a general framework for workload adaptation in autonomic DBMSs, which is 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*.

We then implement a query scheduler that performs workload adaptation in a DBMS, as the test bed to prove the effectiveness of the framework. The query scheduler 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. First-principle performance models are used to predict the performance under the new resource allocation plan. Experiments with IBM® DB2® are conducted to show the effectiveness of the framework.

The effectiveness of the workload adaptation depends on the accuracy of the performance prediction. Finally we introduce a tracking filter (Kalman filter) to improve the accuracy of the performance prediction. Experimental results show that the approach is able to reduce the number of unpredicted SLO violations and prediction errors.

Co-Authorship

Paper “B. Niu, P. Martin, W. Powley, R. Horman, and P. Bird, “Workload Adaptation in Autonomic DBMs”, *Proceedings of CASCON 2006*, Toronto, Canada, October 16 – 19, 2006, pp 161 - 173” is based on Chapters 3 and 4, and Section 5.1.

Paper “B. Niu, P. Martin, W. Powley, P. Bird, and R. Horman, “Adapting Mixed Workloads to Meet SLOs in Autonomic DBMSs”, *Proceedings of the 2nd International Workshop on Self-Managing Database Systems (SMDB 2007) in Conjunction with the 23rd International Conference on Data Engineering (ICDE 2007)*, Istanbul, Turkey, April 15 - 20, 2007” is based on Chapters 3 and 4, and Section 5.2.

Paper “B. Niu, P. Martin, W. Powley, and P. Bird, “Quantifying Workload Importance”, *Fourth International Workshop on Engineering Autonomic Software Systems*, Toronto, Canada, October 23 – 24, 2007” is based on Section 4.2.2.

Acknowledgements

This research would not have been possible without the contributions, help, and support of many people. First, I would like to thank my supervisor Dr. Patrick Martin for his great assistance throughout the course of my study. I would also like to thank Wendy Powley and the rest of the members of Database Systems Laboratory at Queen's University for their support, opinions, and friendship. Additional thanks to Dr. Nicolas Graham and Dr. Juergen Dingel for their suggestions and guidance with this research.

It was a huge help to my research to work with the talented and bright people at IBM. I would like to thank Paul Bird and Randy Horman for all their help, ideas, patience and support. A special thanks to Keith McDonald and other IBM members for helping me obtain information and answering my questions.

I would also like to thank Queen's University, 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 for their strong financial support.

Finally, I would like to thank my wife, my daughter and family for their endless love, support, encouragement and understanding.

Statement of Originality

I hereby certify that all of the work described within this thesis is the original work of the author. Any published (or unpublished) ideas and/or techniques from the work of others are fully acknowledged in accordance with the standard referencing practices.

Baoning Niu

January, 2008

Table of Contents

List of Figures

List of Tables

List of Symbols

α	Importance factor
a_i^k	The slope of the i^{th} process equation at the k^{th} step
$\mathbf{A_k}$	Process matrix at the k^{th} step
C	The total cost limit allowed for all service classes.
C^i	The class cost limit of a service class at the i^{th} control interval
C_i^k	The class cost limit of the i^{th} service class at the k^{th} control interval
f	System objective function
f_i	The utility function of the i^{th} service class
g	The real performance of a service class
$\bar{g}$	The performance goal of a service class to be achieved
$\hat{g}$	The worst performance allowed for a service class
g_i	The predicted performance of the i^{th} service classes
$\bar{g}_i$	The performance goal of the i^{th} service class to be achieved
$\hat{g}_i$	The worst performance allowed for the i^{th} service class
g_i^k	The performance of the i^{th} service class at the k^{th} control interval

$\mathbf{H_k}$ Measurement matrix at the k^{th} step

h_{ij}^{k} The j^{th} slope of the i^{th} measurement equation at the k^{th} step

$\mathbf{I}$ Identity matrix

$\mathbf{K}_k$ Kalman gain at the k^{th} step

l The number of terminal users in a service class

L The number of control intervals for calculating the process error covariance

m The importance level of a service class

m_i The importance level of the i^{th} service class

N The number of queries measured in a control interval

N^* The number of queries needed to reach the predefined measurement accuracy of 95% confidence interval of ±5% measurement

$\hat{\mathbf{P}}_k$ The a prior estimation error covariance at the k^{th} step

$\tilde{\mathbf{P}}_k$ The a posteriori estimation error covariance at the k^{th} step

q_{ii} The i^{th} diagonal element of process noise covariance matrix $\mathbf{Q}_k$

r_{ii} The i^{th} diagonal element of measurement noise covariance matrix $\mathbf{R}_k$

$\mathbf{Q}_k$ The covariance matrices of the process noise at the k^{th} step

r_i The amount of resources allocated to the i^{th} service class

$\mathbf{R}_k$	The covariance matrices of the measurement noise at the k^{th} step
T^i	The average response time of queries in a service class at the i^{th} control interval
t^k	The average response times of an OLTP class at the k^{th} control interval
T_r	Response time
T_s	Service time, the time for doing the actual work
T_t	Average thinking time
T_w	wait time
u_i	The utility of the i^{th} service class
u_i^k	The utility of the i^{th} service class at the k^{th} control interval.
V^i	The query velocity of a service class at the i^{th} control interval
v_i^k	The measurement noise at the k^{th} step in the i^{th} measurement equation
$\mathbf{v}_k$	Measurement noise matrix at the k^{th} step
W^i	The average wait time of queries in a service class at the i^{th} control interval
w_i^k	The process noise at the k^{th} step in the i^{th} process equation
$\mathbf{w}_k$	Process noise matrix at the k^{th} step
X	Throughput
x_i	The i^{th} state variable

$\mathbf{x}_k$ State matrix at the k^{th} step

$\hat{\mathbf{x}}_k$ Estimated state at the k^{th} step

$\tilde{\mathbf{x}}_k$ Predicted state at the k^{th} step

y_i The i^{th} measurement variable

$\mathbf{Y}_k$ Measurement variable matrix at the k^{th} step

$\mathbf{z}_k$ Innovate at the k^{th} step

λ Lagrange multiplier

τ The average service time of queries in a service class

List of Acronyms

API	Application Programming Interface
ASM	Active System Management
DBMS	Data Base Management System
DSS	Decision Support System
IP	Internet Protocol
ISA	Industry Standard Architecture
MPL	Multiprogramming Level
OLAP	On-Line Analytical Processing
OLTP	On-Line Transaction Processing
PCI	Peripheral Component Interconnect
PI	Performance Index
QoS	Quality of Service
QP	Query Patroller
QPNP	Query Patroller without Priority Control
QS	Query Scheduler
RMS	Root Mean Square

SCSI	Small Computer System Interface
SLG	Service Level Gaol
SLO	Service Level Objective
TCO	Total Cost of Ownership
TCP	Transmission Control Protocol
TPC	Transaction Processing Performance Council
WLM	Workload Manager

Chapter 1

Introduction

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 basic idea behind workload adaptation is that by filtering and controlling work requests from different applications against DBMSs, the resources allocated to the applications can be adjusted accordingly and their SLOs brought under control. For example, if a DBMS is experiencing a heavy load from a less important application, it could delay the queries from that application in order to allow queries from more important applications to meet their performance goals. Workload adaptation does not directly deal with resource allocation, and therefore, it does not require low-level resource control 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.

Performance management for DBMSs is becoming increasingly important to businesses as their workloads become more diverse and complex. The emerging trend of server consolidation [HP03] has led to an environment with increased competition for

shared resources between applications from potentially disjoint organizations in a single instance of a DBMS. This results in a workload with diverse and dynamic resource demands and with competing performance objectives for the applications. In addition, Web-based applications introduce a need for flexible and guaranteed application service levels because they tend to involve unpredictable workloads, and a high rate of overall growth in workload size [DHBA02].

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 in a consolidated server. 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 can vary widely across applications. 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. Finally, the complex resource allocation must be done on the fly, which calls for autonomic computing techniques.

Autonomic computing [GC03] is intended to simplify management complexity by providing self-configuring, self-healing, self-optimizing and self-protecting systems. By incorporating autonomic techniques into the performance management process, DBMSs can better deal with the complexity of the management.

The goal of our research is to solve the resource allocation problem through using autonomic computing techniques. We define a general framework for workload adaptation for DBMSs and present a set of key techniques that are necessary to implement the framework. Under this framework a DBMS is able to manage multiple classes of queries to meet their performance goals by allocating DBMS resources through admission control in the presence of workload fluctuation. To achieve this objective, the system detects workload changes and controls the workload on a regular basis. The set of techniques used for these purposes includes workload characterization, performance modeling, workload control and system monitoring. This research discusses these techniques and focuses on their use for tasks such as performance prediction and resource allocation.

The research makes several contributions to realizing 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. 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 an objective function, which encapsulates the performance goals of all classes and their importance to the business. A performance model is used to predict the performance of the DBMS under the new resource allocation plan.

The third contribution is a cost-based performance model, which is a queuing network model [LZGS84] using query cost as resource demand. Under this model, different performance goals, such as response time and execution velocity can be used. A performance model is used to predict the performance of a resource allocation plan, so that an optimal plan can be picked up from the plan space.

The fourth contribution is the introduction of tracking techniques to improve the accuracy of performance prediction. The tracking filter is a Kalman filter [Kalm60, WB01], which strikes a balance between the predictions made by the performance model and the actual measured performance.

The rest of the thesis is structured as follows. Chapter 2 reviews the literature of workload adaptation. A brief history of workload management is first presented and then followed by a discussion of the state of the art of workload management. Chapter 3 explains the general framework for workload adaptation. Chapter 4 discusses the Query Scheduler, which is a prototype implementation of the framework. The key techniques for the framework of workload adaptation are discussed. The evaluation of Query Scheduler is provided in Chapter 5. The further improvement of the accuracy of

performance prediction using Kalman Filters is discussed and evaluated in Chapter 6. We conclude and suggest future work in Chapter 7.

Chapter 2

Literature Review

Workload adaptation is an autonomic computing technique for workload management, in which workload control is performed automatically in a timely manner when workload changes cause performance degradation and inefficient use of resources. To better understand the state of the art techniques for workload adaptation, we first introduce the concept of autonomic computing in Section 2.1, and present a brief introduction of workload management in Section 2.2. We then discuss the state of the art of workload adaptation in Section 2.3.

2.1 Autonomic Computing

Autonomic computing is the discipline with the mission to simplify environment and infrastructure complexity by governing all computations within a given system [Ange02]. The concept was introduced by Paul Horn, senior vice president of IBM Research, in 2001 [GC03].

The drivers for autonomic computing are from both industry and the marketplace. In industry, “the spiraling cost of managing the increasing complexity of computing systems is becoming a significant inhibitor that threatens to undermine the future growth

and societal benefits of information technology" [GC03]. In the marketplace, there is an increasing management focus on "return on investment" while human costs exceed the technology costs. To address the problems, Horn suggested developing autonomic computing systems that regulate themselves much in the same way our autonomic nervous system regulates and protects our bodies.

There is also another reason why autonomic computing is necessary for workload management. The dynamic and diversified workloads cannot be manipulated consistently and effectively by human operators. Human operators allocate resources to workloads by predicting the trends of workloads when the workloads are relatively stable. When workloads change in a fast and unpredictable way, allocating resources in a timely manner is beyond the human operators' abilities.

Autonomic computing systems have four fundamental characteristics [GC03]: self-configuring, self-healing, self-optimizing and self-protecting. Self-configuring means systems can automatically adapt to dynamically changing environments. New features, applications and servers can be dynamically added to the enterprise infrastructure with no disruption of services. Self-healing is the ability of systems to discover, diagnose and react to disruptions. Such a system needs to be able to predict problems and take necessary actions to prevent the failures from having impact on services. For a system to be self-optimizing, it should monitor itself and tune resources automatically to maximize resource utilization to meet users' performance requirements. Self-protecting means systems are able to anticipate, detect, identify, and protect themselves from attacks from anywhere.

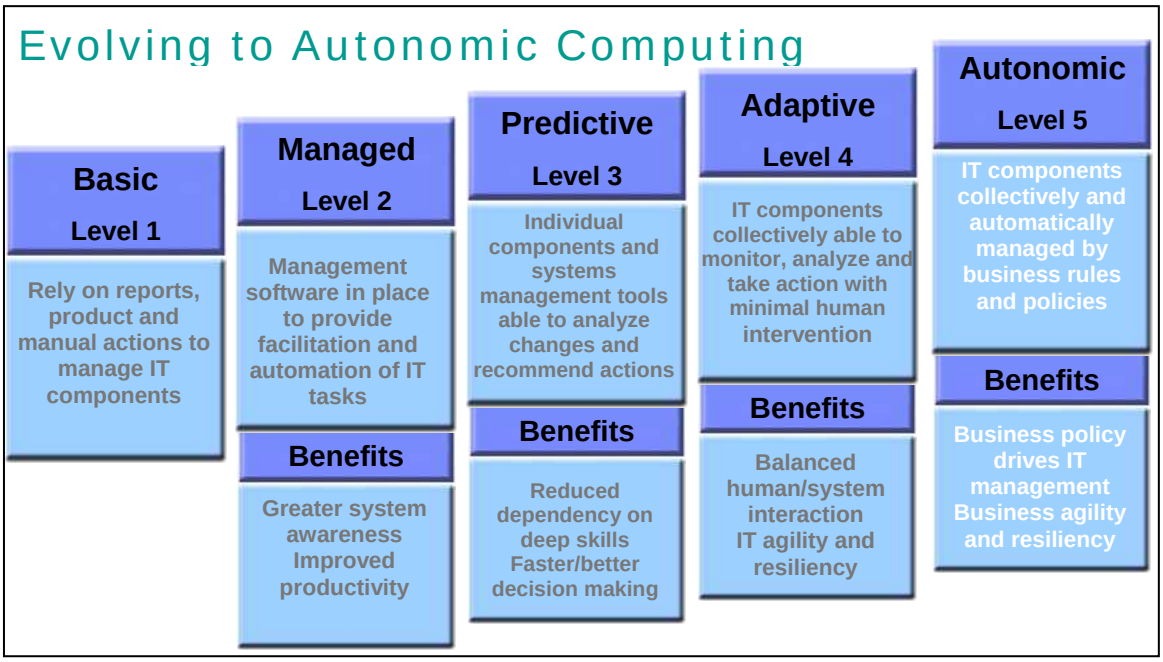

Figure 1 Roadmap of autonomic computing [Ligh04]

To implement autonomic computing, the industry must take an evolutionary approach and deliver improvements to current systems that will provide significant self-managing value to customers without requiring them to completely replace their current IT environments [GC03]. Figure 1 shows the evolution roadmap of autonomic computing. Most commercial products are at the managed or predictive levels.

At the basic level, IT components are manually managed by IT professionals. At the managed level, systems management technologies are used to collect information from systems, which reduce the time to collect and synthesize information. At the predictive level, as new technologies are introduced that provide correlation among several components of the system, the system itself can begin to recognize patterns, predict the optimal configuration, and provide advice on what course of action the

administrator should take. At the adaptive level, systems themselves can automatically take the correct actions based on the information that is available to them and the knowledge of what is happening in the systems. Finally, at the fully autonomic level, the system operation is governed by business policies and objectives. Users interact with the system to monitor the business processes or alter the objectives [Ligh04] [GC03].

2.2 Workload Management

Workload management is concerned with three entities: workload, management objectives, and computing resources. The relationships among them are shown in Figure 2. Whether the management objectives can be achieved or not is determined by the amount of resources allocated to the workload. If the workload gets its desired resources, the management objectives can be achieved, otherwise, they cannot. Different types of workload management take different approaches to how resources are allocated or managed.

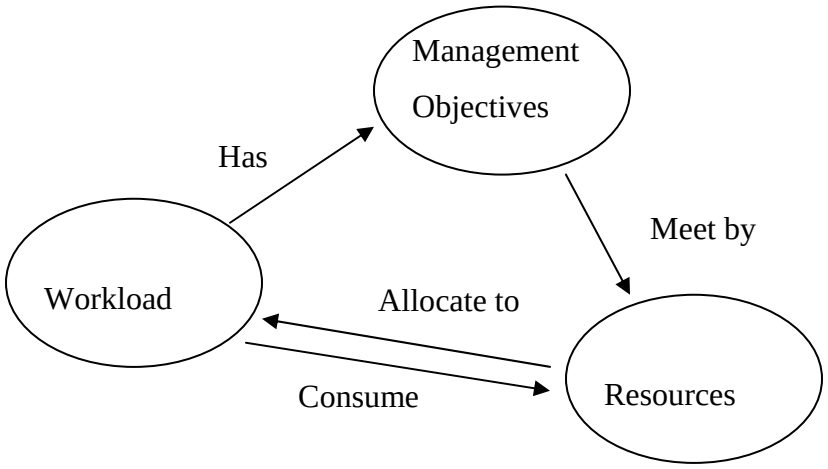

Figure 2 Relationships among workload, management objective and resources

Workload management has evolved mainly from off-line analysis in the early days, focusing on workload characterization and performance modeling, to on-line adaptation, emphasizing system monitoring and workload control. The main impetus is the progress of computer technologies, especially the evolution of computer systems, which continues to offer new capabilities and solutions. To better understand how workload management has reached its current level of complexity such that autonomic properties of workload management are urgently needed, it is imperative to understand how the progress of technologies has changed the objectives of workload management.

In the early days, mainframes were the only computer systems to gain wide acceptance in commercial areas. The strengths of mainframes, such as power, high utilization rate, multiprogramming, and well defined processes and procedures, made them valuable to large-scale data processing or enterprise computing [HP03]. Because of the high cost involved in purchasing, deploying and maintaining them, it was necessary that the mainframes be able to run multiple applications to amortize the cost, which introduced time-sharing systems. At this time, the earliest workload management was born. The primary objective was cost sharing. The management objective was to maintain system availability. SLOs were defined as fairness among applications.

Deploying a mainframe was a time consuming task and it took years to produce the system and develop the applications. Business changes over time, thus the workload at the time of purchase might differ from the workload at the time the system is put into production. Capacity planning was introduced to project budget requirements prior to purchase, and to predict the future workload requirements.

Capacity planning is a process to ensure that the processing capacity can, and always will, deliver the services needed to meet the productivity goals within their budget requirement [LD86]. It is concerned with the management of resources, system configuration, and the prediction of service impact due to workload changes. The primary means to carry out capacity planning are workload characterization and performance modeling, which are part of workload management. Although capacity planning and workload management are not inclusive of each other, the main processes overlap. Historically, workload management was part of capacity planning.

During the 1970s and 1980s, minicomputers became an attractive alternative to mainframes. They were much smaller than mainframes, were much less expensive, and gave developers more freedom in terms of the rules and processes used. In many ways, minicomputers were the first step towards freedom from mainframe computing [HP03]. The low cost of minicomputers relaxed the need for capacity planning, because buying a new minicomputer was not a strain on the budget. The weak processing power of minicomputers prompted the need for better resource utilization in order to deliver a good level of service. In the late 1980's, research in the area of capacity planning was sparse. Workload management was gradually entering into a new phase: resource-oriented workload management, a process to allocate resources to different applications by maximizing the resource utilization to maintain service levels. The popularity of the UNIX operating system accelerated this trend [HP03].

The widespread use of UNIX made applications portable and gave businesses the freedom to choose their hardware and software, thus dramatically decreasing the cost and

time to deploy new applications. UNIX also is the origin of modern computer networks, which in turn heralded the birth of distributed computing systems. The emergence of distributed computing scaled up processing power and in some ways, simplified the process of capacity planning. When capacity is in short supply, or when new applications are introduced, the solution might be simply adding an inexpensive desktop server.

Distributed computing, however, introduced new challenges in the form of more complex resource management and system management in general. Applications became more mission-critical and desktops were used as data processing servers. The number of servers grew, which made the job of managing this disparate environment increasingly complex. Lower service levels and higher service costs usually resulted from increased complexity. To reverse this process, server consolidation began [HP03].

Server consolidation involves running more than one application in a single instance of the operating system. The motivation is that reducing the number of servers simplifies the infrastructure, and improves efficiency and consistency, thereby, reducing the total cost of ownership (TCO). Centralization once again became fashionable. The difference is that, in the past, the most urgent priority for a business was to get their business applications up and running [HP03], but now much more attention is placed on the performance of the applications.

Server consolidation results in a workload with diverse and dynamic resource demands and service level objectives. Web-based applications also introduce a need for flexible and guaranteed application service levels, because they tend to involve widely

unpredictable workloads, with a high rate of overall growth in workload size [DHBA02]. This requires workload management to be business objective oriented and to provide efficient differentiated service at fine granularity to maintain high utilization of resources with low management costs. Resource-oriented workload management has become unable to meet the complex business objective oriented performance requirements. Instead, performance-oriented workload management is required.

Performance-oriented workload management involves monitoring and allocating data processing resources to applications according to SLOs or informal objectives. It involves an ongoing cycle of measuring, planning, and modifying [IBM04]. The goal of performance-oriented workload management is to make the best use of available resources and to meet the objectives without an excessive tuning effort. Unlike capacity planning, which is characterized as off-line analysis, and resource-oriented workload management that features predefined resource constraints, performance-oriented workload management involves on-going workload characterization, continuous monitoring, and control based on business objectives or policies, which calls for little human intervention. Such a system or a process is self-managing since it can adapt to workload changes by performing self-configuration and self-optimization. This adaptive workload management is commonly known as autonomic workload management [Ligh04].

To sum up, workload management has evolved through three phases, from its infancy as a means for capacity planning, focusing on cost sharing, to resource-oriented workload management, with the primary goal being resource utilization, and finally to

today's performance-oriented workload management, with a focus on business objectives. This evolution has been driven by the advancement of technologies and has as its basis, cost sharing and SLOs. Its focus has shifted from cost sharing to performance; its style has changed from off-line analysis to on-line adaptation.

2.3 The State of the Art of Workload Adaptation

Workload adaptation, as an autonomic technique for workload management, is a performance-oriented workload management technique. The most important feature of workload adaptation techniques is their adaptability: how systems detect workload changes and react to the changes by controlling the workload. In this section we discuss the state of the art of workload adaptation techniques, focusing on the mechanisms by which they adapt to workload changes.

2.3.1 Adaptive Mechanisms

The area of workload adaptation in DBMSs has been examined by a number of researchers. Brown et al. [BMCL94] 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 adjustment mechanism takes a heuristic approach. Each class has its heuristics for calculating its MPL and memory allocation parameters. Workload changes are detected by monitoring performance data. For simplicity, the heuristics disregard the interference between classes. The interdependency between

classes that results from the competition for shared resources is solved by performance feedback.

Pang et al. [PCL95] 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. Workload changes are detected by constantly monitoring the resource demand. A workload change causes the system to calculate a new MPL, reallocate memory and adjust priorities. The new MPL is calculated by applying two techniques: a miss ratio projection and resource utilization heuristics. The miss ratio projection uses a least squares fit to produce a quadratic equation based on the previous observations of the MPL and miss ratio. The current miss ratio is then projected to a new MPL based on the equation. The resource utilization heuristic tries to minimize the miss ratio by keeping the resource utilization within a healthy range. The system divides queries into two priority groups, called a regular group and a reserve group, and assigns each query class a quota for the regular group. Queries in the regular group are assigned a priority based on their time deadlines, while queries in the reserve group are assigned a priority lower than those in the regular group. This ensures the queries in the regular group are admitted and get resources first. The miss ratio distribution is maintained by adjusting the regular group quota for each class.

Both Brown et al. and Pang et al. use heuristics to determine new workload control plans. Performance objectives are dealt with individually. While the predefined heuristics are simple to implement, they disregard system dynamics and resource tuning is at a coarse level. In our study, we use performance objective encapsulation techniques to manage multiple classes collectively to reflect the interdependence between classes, and take a performance model based approach to find a solution for workload control to reflect the system dynamics, hence, fine tuning resource allocation.

Commercial systems currently support resource-oriented workload control, despite the fact that they have begun to introduce performance objectives as the secondary management objective. Teradata's Active System Management [Ball02] (ASM) controls the workload presented to a DBMS by using predefined rules based on thresholds of the workload such as MPLs and number of users. Workload classification is the basis for managing resources and translating performance goals into resource requirements.

ASM uses workload definition to define a performance class based on the properties of queries and how they are managed. A workload definition is a tuple <classification rule, MPL, exception, service level goal>. Classification rules define the attributes of the request that qualify the query to run under this Workload Definition. MPL defines how many queries can be running at one time under the Workload Definition. When the threshold is exceeded, new queries are placed on the delay queue. Exceptions trigger workload control actions and are produced when certain abnormal characteristics such as high skew or too much CPU processing are detected after a query

begins execution. Service level goals (SLG) specify the objective to be achieved for the workload class.

ASM does not predict resource demand for each performance class. It maps a performance class to an allocation group with a fixed weight. Allocation groups compete for resources based on their weight. A performance class can have several periods and maps to different allocation groups in different periods. Administrators make the assignments based on their understanding about the workload. ASM takes a preventive approach to workload changes. It reacts to the exceptions defined in the workload definition with admission control. Resource reallocation can be triggered when the resources allocated to an allocation group are under utilized.

Since ASM allocates resources to allocation groups, resource allocation might not reflect the requirements of all the performance classes mapped into the allocation groups. In our study, resources are directly allocated to the service classes in need and the SLOs can be directly addressed.

DB2 Query Patroller (QP) [IBM03B] uses estimated query costs and MPLs to perform admission control. It can dynamically control the admission of queries against DB2 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, threshold management, and submitter queue prioritization. A query class is defined by specifying a cost range and an MPL threshold. Queries are

assigned to a query class based on the cost of the 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 with potentially different MPL thresholds and so uses system resources more effectively. 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. 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. 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. [MAFM99] [MB03] [MBD01] 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. As a general approach for workload management, QoS Controller deals with three performance goals, namely average response time, average throughput and probability of rejection, and manages them collectively.

A QoS metric is devised to encapsulate all the performance goals and strikes a balance among them. QoS is defined as the weighted sum of relative deviations of the

three performance metrics with respect to their desired goals. QoS Controller takes a performance model based approach to adapt to the workload changes. A queuing model is used to predict the performance for maximizing the QoS metric. This approach requires that QoS Controller is able to acquire both workload characteristics and actual performance. QoS Controller analyzes the workload and computes statistics for the workload, such as average arrival rate, at regular intervals and uses statistical techniques to forecast the intensity of the workload in the next interval. The current and predicted workload intensity values and the measured performance data are used as input parameters of a queuing model to predict the performance for the next interval. This approach requires no human intervention and automatically adapts to workload changes.

Pacifici et al. [PSTY05] 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.

Both Menascé and Pacifici 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.

2.3.2 Comparison of the State of the Art Techniques

We have investigated workload adaptation techniques in terms of their adaptability. We summarize our observations below and in Table 1. For ease of presentation, we name the techniques as follow:

M & M	Brown's algorithm
PAQRS	Pang's algorithm
ASM	Teradata's Active System Management
DB2 QP	DB2 Query Patroller
QoS Contorller	Menascé's Quality of Service (QoS) Controller
WSWLM	Pacifici's performance management system

Except for QoS Controller, all other workload adaptation mechanisms use classification rules to partition workloads into classes. Two of them (M & M, DB2 QP) use resource demand as a criteria, and three of them (WSWLM, PAQRS, ASM) use performance goals. Workload classification based on resource demand makes it easy to control resource allocation, while workload classification based on performance goals makes performance tracking easy. A two level classification based on both criteria is desirable for translating high level performance goals into low level resource allocation, which involves both performance tracking and resource allocation.

There are two approaches to detecting workload changes: a workload characterization approach (PAQRS) and a performance monitoring approach (QoS Controller, WSWLM, M & M, ASM). A workload characterization approach tracks workload changes before performance is affected and makes proactive workload control possible. A performance monitoring approach, on the other hand, always takes control actions after workload performance degrades.

The three approaches for solving performance problems used in the systems are a performance model approach (QoS Controller, WSWLM), a heuristic approach (M & M, PAQRS), and threshold control (DB2 QP, ASM). A performance model approach is the best approach to adapt workload changes because it can catch system characteristics at a finer granularity than the others approaches. Both a heuristic approach and a threshold approach solve performance problems at a much coarser granularity than the performance model approaches.

There are three ways to define the control interval at which workload management systems periodically adapt to workload changes: regular time interval (QoS Controller, WSWLM), number of completions of work requests (PAQRS), and performance goal violation (M & M, ASM). As we can see from Table 1, none of the techniques discussed reaches the proactive level that requires that both workload changes are detected through workload characterization and performance is predicted through an evolving performance model.

Table 1 Comparison of workload adaptation techniques

Systems	Management Objective	Adaptability	System awareness	Workload Control Methods
QoS Controller	**Service availability**: response time, throughput and probability of rejection encapsulated in QoS metric.	**Reactive level**: Classification: no; Detection: performance monitoring; Solving: performance model; Interval: regular time.	**Closed-feedback**: performance and resource usage metrics.	**Parameter tuning**: optimization techniques guided by performance model.
WSWLM	**Service availability**: average response time encapsulated in utility functions.	**Reactive level**: Classification: performance goal based; Detection: performance monitoring; Solving: performance model; Interval: regular time.	**Closed-feedback**: performance metrics.	**Admission control**-MPL: optimization techniques guided by performance model.
M & M	**Service availability**: average response time goal.	**Assisted level**: Classification: performance goal based; Detection: performance monitoring; Solving: heuristic; Interval: goal violation.	**Closed-feedback**: performance metrics.	**Resource allocation**: heuristics **Admission control**-MPL: heuristics, non-integral MPL
PAQRS	**Service availability**: time deadline with miss predefined distribution.	**Assisted level**: Classification: performance goal based; Detection: workload characterization; Solving: heuristic; Interval: number of completions.	**Closed-feedback**: performance and resource usage metrics.	**Parameter tuning**: heuristics **Resource allocation**: heuristics **Admission control**-MPL: heuristics
DB2 QP	**System availability**: keep DB2 in safe operation region.	**Manual level**: Classification: resource demand based; Detection: workload characterization; Solving: thresholds; Interval: no.	**Report**: performance and resource usage metrics	**Admission control**: MPL and cost set by administrators
ASM	**Service availability**: response time, throughput, and percentile response time associate with a weight.	**Assisted level**: Classification: performance goal based; Detection: performance monitoring; Solving: thresholds; Interval: goal violation.	**Closed-feedback and report**: performance and resource usage metrics.	**Resource allocation**: period aging, relative weight **Admission control**: workload aggregates.

We note that the most desirable workload characterization techniques are workload classification and statistics. Workload classification based on predefined classification rules, however, is not suitable for a fast changing workload. An unsupervised partitioning technique, such as clustering, is a better choice. Clustering groups work requests into classes such that the work requests in a class are similar to each other and dissimilar to the work requests in other classes.

2.3.3 Component Specific Techniques

Component specific techniques for workload adaptation mainly focus on workload characterization and performance modeling. There exists a large body of research on clustering [XW05] [JMF99] [Berk02] [JD88]. Various algorithms have been proposed for generalized clustering [Ande73] [DHS01] [DO74] [ELL01 [Hart75] [JD88] [XW05] and domain specific clustering [NH94] [JTZ04] [He99]. Workload adaptation calls for light-weight and online clustering techniques, such as incremental clustering. Incremental clustering techniques handle work requests one at a time and do not require that the entire data set be stored. Typically, they are non-iterative, so the time requirements are minimal [JMF99]. Most incremental clustering algorithms are dependent on the order of the input pattern [Moor89]. Some typical incremental clustering algorithms are presented in [CG90], [SCH75], [Fish87], [Can93] and [CCFM04].

There also is a large body of research in the area of performance modeling. Much work has been done in the area of queuing network theory [LZGS84] [Jain91] [MA98]. Woodside [Wood02] went one step further and proposed layered queuing network

models [Wood89] to adapt the queuing network model to complex computer systems, particularly for software design [ECW01] [ZW03] [XWP03]. With the emergence of autonomic computing, various on-line performance models [MB03] [CA02] have been proposed. To adapt to rapidly changing systems, Diao et al [DEFH03] developed a framework for discovering performance models on-line. Evolvable performance models require that the system be able to track changes of the model parameters such as resource demands, cluster centroids, etc.

Relevant workload control tricks found in the literature include the following:

Non-integer MPL: Admission control is usually performed at the granularity of a single query. Non-integer MPL can help tune the admission control at an even finer granularity. Non-integer MPL is produced by first finding the lowest integer MPL at which performance goals are exceeded, and then delaying the admission of the next query by an amount of time that is equal to the amount by which the previous query exceeded its goal. No delay is used if the previous query violated its goal. By delaying the admission of a new query, the average actual MPL is forced to be some fraction lower than the integer MPL. In effect, the delay makes the system behave as if each query's response time exactly equals the goal for the class [BMCL94].

Period aging mechanism: In order to avoid long running work requests consuming too many resources, IBM z/OS WLM [IBM03A] uses a period aging mechanism to control resources assigned to them without knowing their resource demand. The period aging mechanism runs work requests in periods. In each period, a work request can

consume a certain amount of resources. When the work request consumes its amount of resources for the current period, it enters the next period with less resources and lower importance. In this way short running work requests are honored. As shown in Figure 3, a work request is assigned 50% of CPU time during the first period with a high importance

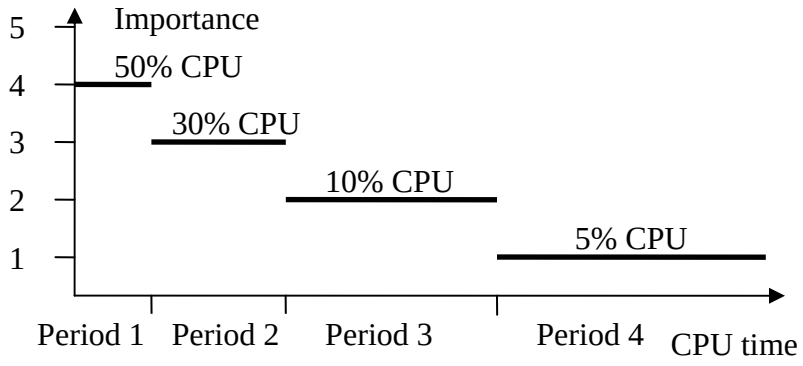

Figure 3 Period aging mechanism

level 4. If the work request continue running into Period 2, it is assigned 30% of CPU time with a lower importance level 3, and 10% of CPU time with importance level 2 in Period 3 and finally 5% of CPU time with importance level 1.

2.4 Summary

Workload adaptation is an autonomic computing technique for workload management. The four aspects of autonomic computing: self-configuring, self-healing, self-optimizing and self-protecting, can be condensed to one word, that is adaptive. That is to say, an autonomic computing system or technique must be able to predict and react to any changes in order to keep the system running in a healthy operation region.

Workload management is concerned with three entities, namely the workload to be managed, the objectives to be achieved and the resource to be manipulated. Workload management has evolved through three phases, from the means for capacity planning, focusing on cost sharing, to resource-oriented workload management, with the primary goal being resource utilization, and finally to today's performance-oriented workload management, with a focus on business objectives. Its style has changed from offline analysis to online adaptation.

We have discussed the state of the art of workload adaptation. Workload adaptation mechanism consists of workload detection and workload control. There are two approaches to detecting workload changes: a workload characterization approach and a performance monitoring approach. A workload characterization approach tracks workload changes before performance is affected and makes proactive workload control possible. A performance monitoring approach, on the other hand, always takes control actions after workload performance degrades.

There are three approaches for deriving workload control plan, namely, a performance model approach, a heuristic approach, and a threshold approach. A performance model approach is the best approach to adapt to workload changes because it can catch system characteristics at a finer granularity than the others approaches. Both a heuristic approach and a threshold approach solve performance problems at a much coarser granularity than the other approaches.

We note that workload adaptation calls for light weight and online clustering techniques for workload characterization, and evolvable performance models for deriving workload control plans.

Among the adaptive mechanisms discussed, some are limited to the number of service classes can be handled, and others cannot handle business importance. Most of the mechanisms are heuristic based approaches. The difficulties for workload adaptation are how to manage multiple SLOs with different business importance, how to adapt to the workload changes while meeting the SLOs and how to prediction performance accurately. We will present a framework for workload adaptation to address these difficulties in Chapter 3.

Chapter 3

General Framework for Workload Adaptation

Generally speaking, the ultimate objectives of workload management are cost sharing and meeting SLOs. Cost sharing can be achieved by minimizing administrative cost and maximizing resource utilization. Minimizing administrative cost means minimizing human intervention and allowing systems to manage themselves automatically. Maximizing resource utilization requires efficient use of available computing resources to reduce the need for hardware expenditures. Meeting SLOs means ensuring critical business work is completed so that service requirements are achieved.

The three objectives of minimizing administrative cost, maximizing resource utilization and meeting SLOs may conflict. Meeting SLOs requires sufficient available resources and incurs administrative cost. Maximizing resource utilization compromises SLOs when unbalanced resource requirements are presented by business workloads, and also incurs administrative cost. Prior to autonomic computing, only one or two objectives could be achieved. Resource-oriented workload management [DHBA02] [CTYB01] [MCWV99] aims to maximize resource utilization, and performance-oriented workload management [HP04] [IBM03A] focuses on meeting business goals. With autonomic

computing technologies such as adaptive processing and automatic provisioning of resources, the conflict between the three objectives may be minimized.

To achieve the objectives, workload adaptation, which is one technique for workload management, must be able to detect the fluctuation of the workload (*workload detection process*), and react to the workload changes by controlling the workload (*workload control process*). Note that the word "process" here is not the same concept as the processes in operating systems. It consists of a series of actions or functions applied to underlying messages to bring desired results. We view workload adaptation as consisting of a workload detection process and a workload control process, which, in turn, involve four functional components - *workload characterization*, *performance modeling*, *workload control*, and *system monitoring*. The functional view of workload adaptation is shown in Figure 4.

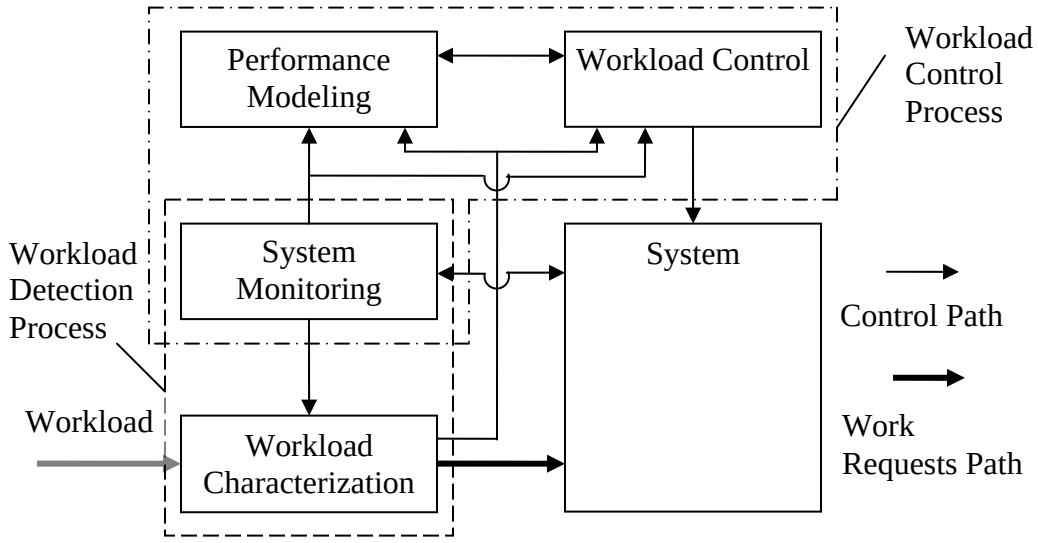

Figure 4 Functional view of workload adaptation

3.1 Functional Components

Workload characterization is concerned with measuring and modeling production workloads [LD86] [MA98]. The purpose of characterizing a workload is to understand and determine the resource usage and performance behavior for subsequent workload control. Most workload characterization performed in workload management is for predicting resource demands as input to performance models that are used to choose a resource allocation plan.

Performance modeling tries to predict the performance of the target system through a model that describes the features of the target system [MA98]. The inputs of 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. Performance models are usually developed off-line. In the autonomic era, systems are self-configurable, which calls for adaptive performance modeling techniques that can evolve performance models in response to changes in the system [DEFH03].

Workload control components find and enforce an optimal workload control plan to meet the management objectives when the fluctuation of workload causes the system performance to degrade. Based on the support of the underlying system, the control plan can be steps to execute direct resource allocation, parameter tuning or admission control. Control actions are triggered by workload changes.

System monitoring, or feedback, tells 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 and helps the workload characterization component to characterize the workload, but also helps the evolution of performance models by tracking changes to systems [IBM03A] [MB03] [BMCL94].

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 4, 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 [MB03] [PSTY05]. 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 provides 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 [MB03] [DEFH03].

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 4).

3.4 Performance Metrics

The primary objective of workload adaptation is to meet SLOs, which are specified by performance metrics. Workload performance metrics provide an overall mechanism for workload management systems to assess how well work is passing through the system and to make adjustments accordingly. They should allow workload management systems to examine the workload for its importance and user defined service objectives, and evaluate whether the system is meeting those objectives. Usually three types of performance metrics [Adam01] are used for workload management systems, namely performance goal, workload importance, and performance index.

Performance Goal

Performance goal is the standard for determining whether or not the system is meeting its objectives. The most widely used performance goals are response time, execution velocity, and throughput.

Response time is well understood and defined as:

$$T_r = T_s + T_w$$

Response time T_r consists of service time T_s, which is the time for doing the actual work, and wait time T_w, which is the time spent waiting for resources.

Execution velocity is defined as [IBM03A]:

$$Execution\_Velocity = T_s / (T_s + T_w)$$

A higher velocity goal means that fewer delays are allowed, while a lower velocity goal means that more delays are tolerated.

Throughput (X) is a measurement of the number of committed transactions per unit of time:

$$X = Number\_of\_Transactions / Elapsed\_Time = N / (T_t + T_r)$$

where N is the number of users, T_t is the average think time, and T_r is the average response time. A service objective based on throughput is difficult to attain because the only manageable variable is T_r, the average response time.

How these metrics are used is based on the nature of the workload. When the work requests in a service class behave in a consistent manner then response time is a reasonable measure for its performance goal. Since there may be some variation in response time, it is reasonable to use the response time goal with a percentile violation. Throughput is typically used to measure system capacity. It is commonly combined with response time to define the workload volume at a certain response time level. Execution velocity is used when the response time measure is difficult to define yet the throughput

of the workload is important. For example, velocity can be used for an OLAP (On Line Analysis Processing) workload since the response time varies greatly from one work request to another. In practice, response time goals are assigned to OLTP (On Line Transaction Processing) workload and execution velocity goals are assigned to OLAP workload, because response times for the former are critical, while the waiting time is significant to the latter.

There are two aspects that need to be considered when setting performance goals, namely system capacity and business requirements. A performance goal must be realistic and attainable with the available system resources so that the system operates in a healthy state that is not overloaded and not under used. This can be achieved through experiments to find the reasonable range of a performance goal. After this, a performance goal can be calibrated to a value to meet business requirements.

Workload Importance

Workload importance identifies the order in which service classes receive or donate resources when the system capacity is not sufficient for all service classes to meet their performance goals.

Let us look at a scenario where workload importance comes into play. Consider a system with both OLTP and OLAP workloads where the OLTP workload has higher importance. The relative importance of the workloads has no impact on resource reallocation as long as both workloads can meet their performance goals. Resources should be shifted to the OLTP workload when it violates its performance goal since the

OLTP workload is more important than OLAP workload. If the OLAP workload violates its performance goal, resources should be shifted to the OLAP workload as long as the OLTP workload can also meet its performance goal after this reallocation. When both workloads violate their performance goals, the more important workload, that is the OLTP workload, is given preference.

That is to say, the performance goals guide resource allocation as long as system resources are underutilized and workload importance comes into play when the system capacity is filled. Workload importance is therefore not synonymous with priority in operating systems. First, workload importance is a high level construct defined by users and priority is a low level concept used by the operating system. Second, workload importance only plays a role when a workload is not meeting its goal. Priority, on the other hand, is in effect all the time and operating systems schedule jobs based on their priority. Third, a less important workload may receive resources from a more important workload as long as the latter is meeting its performance goal and the former is failing to meet its performance goal. However, it is not possible for a process with a low priority to take over CPU resources from a process with higher priority.

Performance Index

Performance goals and workload importance define the service objectives to be achieved by workload management systems. The system may or may not be able to achieve the service objectives. Workload management systems should be able to report on system performance and to indicate how close the system is to meeting its service objectives.

Adam [Adam01] uses the performance index (PI) for this purpose. It is calculated by dividing actual performance against the performance goal:

$$PI = Measured\_Performance / Performance\_Goal$$

For a response time goal, if the value is less than 1, the system is exceeding the objective. If it is exactly one, the system is meeting the objective, and if it is greater than one, the system has missed the objective. The PI is not only used to evaluate how well the system is meeting its objectives, it is also useful to determine the accuracy of the defined objectives. There are other ways of calculating performance index, such as the QoS metric defined by Menasce and Bennani [MB03] and the system utility used by Pacifici et. al. [PSTY05].

3.5 Summary

This chapter presents a general framework for workload adaptation. It consists of two processes, namely a workload detection process and a workload control process. The workload detection process detects workload changes and provides knowledge about the workload. The workload control process reacts to the workload changes by performing resource allocation, parameter tuning and admission control to maintain system service levels.

The framework also involves four functional components, namely workload characterization, performance modeling, system monitoring, and workload control. Workload characterization is concerned with measuring and modeling production

workloads. It is the starting point of workload management, which allows the system to manage individual work requests collectively and calculate the resource demand easily, thus simplifying the processes of workload management. Performance modeling tries to predict the performance of the target system through a model that describes the features of the target system. System monitoring, or feedback, tells how well the system is performing by continuously acquiring the execution information of the workload and the resource usage of the system. It is also useful in simplifying the performance model when the system is too complex to analyze and model. Workload control components try to find and enforce an optimal resource allocation plan to meet the management goals.

Three types of performance metrics, namely, performance goal, workload importance, and performance index, usually used for workload adaptation. Performance goals define the objective to be achieved. Workload importance identifies the order in which service classes receive or donate resources. Performance index measures the extent to which the system meeting the specified service objectives.

In Chapter 4, we present a prototype implementation of the framework, called Query Scheduler, for DB2, which adapts the workload for an instance of DB2. 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 effectiveness of the Query Scheduler is evaluated through experiments.

Chapter 4

Query Scheduler

This chapter discusses the implementation of the Query Scheduler, which is a proof-of-concept implementation of the workload adaptation framework discussed in Chapter 3. It automatically controls workloads running on DB2 in order to satisfy defined SLOs. Section 4.1 describes the architecture of Query Scheduler and details the implementation of the two processes defined in our framework, namely the workload detection process and the workload control process. Section 4.2 presents the technique used to encapsulate SLOs. Sections 4.3 to 4.6 discuss the implementation of the four components of our framework, namely workload characterization, performance modeling, system monitor and workload control, respectively.

4.1 The Architecture of Query Scheduler

To implement our framework, we need to define its two processes. The key to defining the workload detection process is acquiring the workload information from DB2, and the main issue for defining the workload control process is determining the workload control methods that DB2 can provide. With this knowledge we can then design the four components of Query Scheduler.

DB2 QP, the workload manager of DB2, provides users with interfaces to intercept queries, acquire query information and, via direct commands, release queries [IBM03B]. When a query comes into DB2, DB2 QP can intercept the query by blocking the agent responsible for the query and collect the query information, including query identification information, query cost and query execution information. Query identification information is useful for query classification. Query cost is expressed in *timerons*, which is a generic measure of resource usage of a query used by the DB2 optimizer to express the combined resource demand to execute the query. We use query cost as the resource demand for a query. Controlling the total cost of all queries in the system or for a service class, we can control the resources used by the system or a service class. Query execution information records the times when a query arrives, starts to execute and completes or aborts, which can be used to calculate the performance of the query. The intercepted queries can be released for execution through DB2 QP's unblocking API.

We make use of DB2 QP's interfaces to define Query Scheduler. The interfaces for intercepting and releasing queries constitute a basic admission control mechanism for the workload control process. The query information acquired by DB2 QP is used for workload detection and making control decisions. As shown in Figure 5, DB2 QP is used by Query Scheduler 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. We modified DB2 QP to inform Query Scheduler each time a query was intercepted.

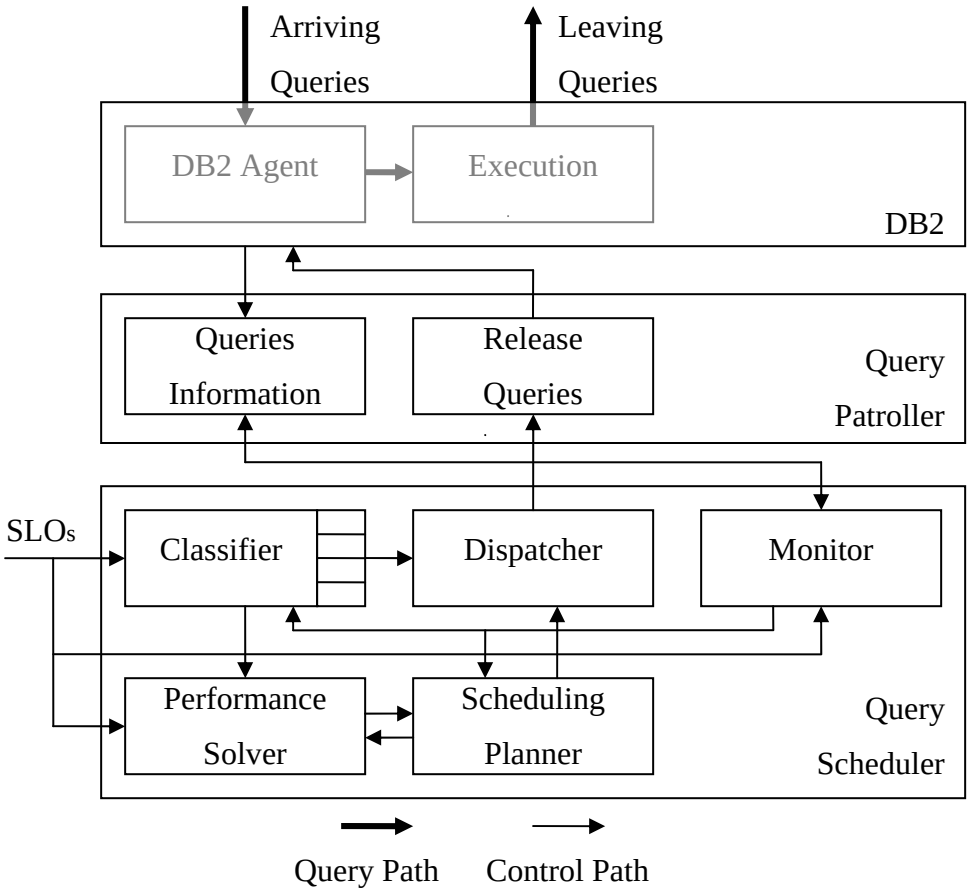

Figure 5 Query Scheduler

The Monitor component then collects the information about the query from the DB2 QP control tables, including query identification information, query cost and query execution information. 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.

The workload control of Query Scheduler is cost-based instead of MPL-based. When queries vary widely in size, the cost-based approach is more reasonable than the MPL-based approach because resource demand measured in terms of cost is more accurate than in terms of MPL. For example, the cost-based approach can admit 10 small queries or 3 big queries with the same total cost, while the MPL-based approach allows 5 queries regardless the query size. 5 big queries might be enough to saturate the system and 5 small queries might leave the system underused. In our implementation a workload control plan is a set of class cost limits. Each service class is assigned a class cost limit expressed in *timerons*, which 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 blocked agents. The Scheduling Planner consults with the Performance Solver at regular intervals to determine an optimal workload control plan, and passes this plan to the Dispatcher.

4.2 SLO Encapsulation

A SLO is often specified by an importance level and a performance goal [Adam01] and is associated with a service class. An importance level describes how important it is to the business for queries to meet the performance goal relative to the other work competing for the same set of system resources. A performance goal defines the desired performance objective.

Managing multiple SLOs is complex. Not only does the complexity increase with the number of SLOs, but also the interdependencies among them makes it even more difficult to control the workload. It is not a trivial problem to translate a high-level performance goal into resource requirements for a single service class. Multiple service classes can interfere with each other's performance through competing for shared resources. Because of the interdependencies, the amount of the control over one service class might result in the performance goal violation for some other service classes. It is desirable to collectively manage all the SLOs by taking into consideration the interdependencies among the classes. In this section we discuss our approach to encapsulating SLOs by using objective functions.

4.2.1 Performance Goals

The most widely used performance goals are response time, throughput, and execution velocity [IBM03A]. 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 is a better choice.

OLTP workloads, such as TPC-C [TPC], consist of small queries with sub-second response time, so an average response time goal is reasonable. OLAP workloads, such as TPC-H, contain queries with widely varying response times. Therefore, a velocity-type goal is most appropriate for OLAP workloads. Motivated by the need to see a clear division between waiting and execution times when performing admission control, we use the metric *Query_Velocity*, which we define as

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

where *Execution_Time* is the amount of time a query runs 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 value between 0 and 1, and is a measure of how fast work is running compared to ideal conditions without delays. It reflects the impact of the workload adaptation mechanisms. A larger value means a shorter waiting time compared with execution time and hence better performance. 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. The total cost limit is a reasonable system saturation threshold and can be obtained through experiments.

4.2.2 Workload Importance

Managing workload importance is similar to managing performance goals of workloads in that both of them translate high level objectives into low level resource allocation plans. The performance of a workload directly relates to the resources allocated to it,

which can be described by a performance model. Enforcing the importance of a workload, however, not only has to do with the resources allocated to the workload itself, but also the other workloads, which is difficult to describe using a performance model.

We also need to answer the question, to what extent does workload importance affect the resource allocation when it comes into play? Generally speaking, there are two interpretations of workload importance when we say one workload is more important than another:

- *Absolute importance*: the requirements of an important workload should be satisfied before those of less important workloads. Important workloads may be allocated all resources and less important workloads must wait.
- *Relative importance*: all workloads share the resources and each workload is assigned the amount of resources commensurate with its importance relative to others.

Absolute importance is relatively easy to implement. The general guideline is to allocate enough resources to meet workload performance goals in the order from the highest importance to the lowest importance until resources are used out or all the performance goals have been met [ABHG07] [SHIN06]. Relative importance, however, is difficult to implement. It not only has to do with the resources allocated to a workload itself, but also the resources allocated to other workloads. In the rest of the thesis, we discuss relative importance.

In our implementation, we use the following process to quantify workload importance.

- Assign an importance sequence number *1* to *n* to the workloads in increasing order of importance.
- Create importance levels by scaling the sequence numbers with the importance freedom *d*, a user defined integer identifying the adjustable range of workload importance.
- Determine α through experiments so that the difference of importance between the two adjacent workloads and the most important workload meet the management expectation.
- Adjust the importance levels of the other workloads in the range $\left[d \bullet i - \frac{d}{2}, d \bullet i + \frac{d}{2} \right]$ so that the resulting importance of each workload meets the management expectation.

In the following sections we discuss how to encapsulate importance levels and performance goals together.

4.2.3 Objective Function

Consider a system with *n* service classes, each assigned a SLO, denoted as $\langle \bar{g}_i, m_i \rangle$, where $\bar{g}_i$ is the performance goal of the i^{th} service class to be achieved and m_i is the importance of the i^{th} service class. We denote $g_1, g_2, ..., g_n$ as the predicted performance of

the service classes given a resource allocation plan $r_1, r_2, ..., r_n$, where r_i is the amount of resources allocated to the i^{th} service class. The performance of the i^{th} service class g_i can be predicted by a performance model $g_i = p_i(r_i)$. The utility [NMPH06] of the i^{th} service class, u_i, which describes how well the system meets its SLO, is a function of $\bar{g}_i, m_i$ and g_i: $u_i = f_i(\bar{g}_i, m_i, g_i)$. Multiple utilities can be encapsulated into an objective function $f(u_1, u_2, ..., u_n)$. The management problem becomes the problem of maximizing the objective function f. By properly choosing the f_i's and f, an optimal workload control plan can be derived to satisfy the management objective. Usually the objective function f is chosen as $f = \sum u_i$, called system utility. Choosing the utility functions, however, is not easy. We discuss how to choose utility functions in the next section.

4.2.4 Utility Function

Utility functions are the key to encapsulating multiple SLOs into an objective function and play an important role in allocating resources among workloads. There is no single way to choose a utility function [PSTY05]. The principles in choosing utility functions are:

- The utility should be larger when the performance experienced by a workload is better than its expected performance goal and vice versa.
- The utility should increase as the performance experienced by a workload increases and decrease otherwise.

- The speed of utility increase should become less as the performance increases and the speed of utility decrease should become greater as the performance decreases.
- The size and shape of the utility function should be controlled by one or two parameters that can be adjusted to reflect the importance of one workload over another [PSTY05].

Based on our experiments we found the following general form of utility function satisfies the above principles:

$$u = 1 - e^{\alpha m \frac{\bar{g} - g}{g - \hat{g}}}$$

where $\bar{g}$ is the performance goal of the service class to be achieved, m is the importance level of the service class, $\hat{g}$ is the worst performance allowed, g is the real performance, and α is an importance factor, which is a constant that can be experimentally determined or adjusted to reflect the extent of the difference between two adjacent importance levels. α controls the size and shape of a utility function.

This format of utility function complies with the above principles and encapsulates both the performance goal and the importance level of a workload. It also has good properties (the second derivative exists) and allows mathematical methods to be used to optimize the objective functions. Besides, it can be applied to both response time goals and execution velocity goals.

Figures 6 and 7 show the utility functions applied to response time goals and to execution velocity goals, respectively. For response time goals, smaller values mean better performance, so the utility functions are a decreasing function. For velocity goals, large values mean better performance, so the utility functions are an increasing function.

From the shape of the utility functions, we observe that as performance improves, the curve becomes flatter. That is to say, when a service class achieves its performance goal, its utility increases slowly as its performance improves (the part of the curves in Figure 6 where g is between 0 and 2 seconds and the part of the curves in Figure 7 where g is greater than 0.5). 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 performance deteriorates, the curve becomes steeper. That is to say that when the service class violates its performance goal, the marginal utility rapidly increases as more resources are assigned to the service class (the part of the curves in Figure 6 where g is greater than 2 seconds and the part of the curves in Figure 7 where g is between 0 and 0.5). In this case, allocating more resources to the service class should bring the class closer to meeting its performance goal.

We also notice that the utility function of a service class with a higher importance level has a steeper curve. In Figure 6 (a), the curve marked with the importance level $m = 2$ is steeper than the curve marked with the importance level $m = 1$. In Figure 7 (a), the curve marked with the importance level $m = 4$ is much steeper than the curve marked with the importance level $m = 1$.

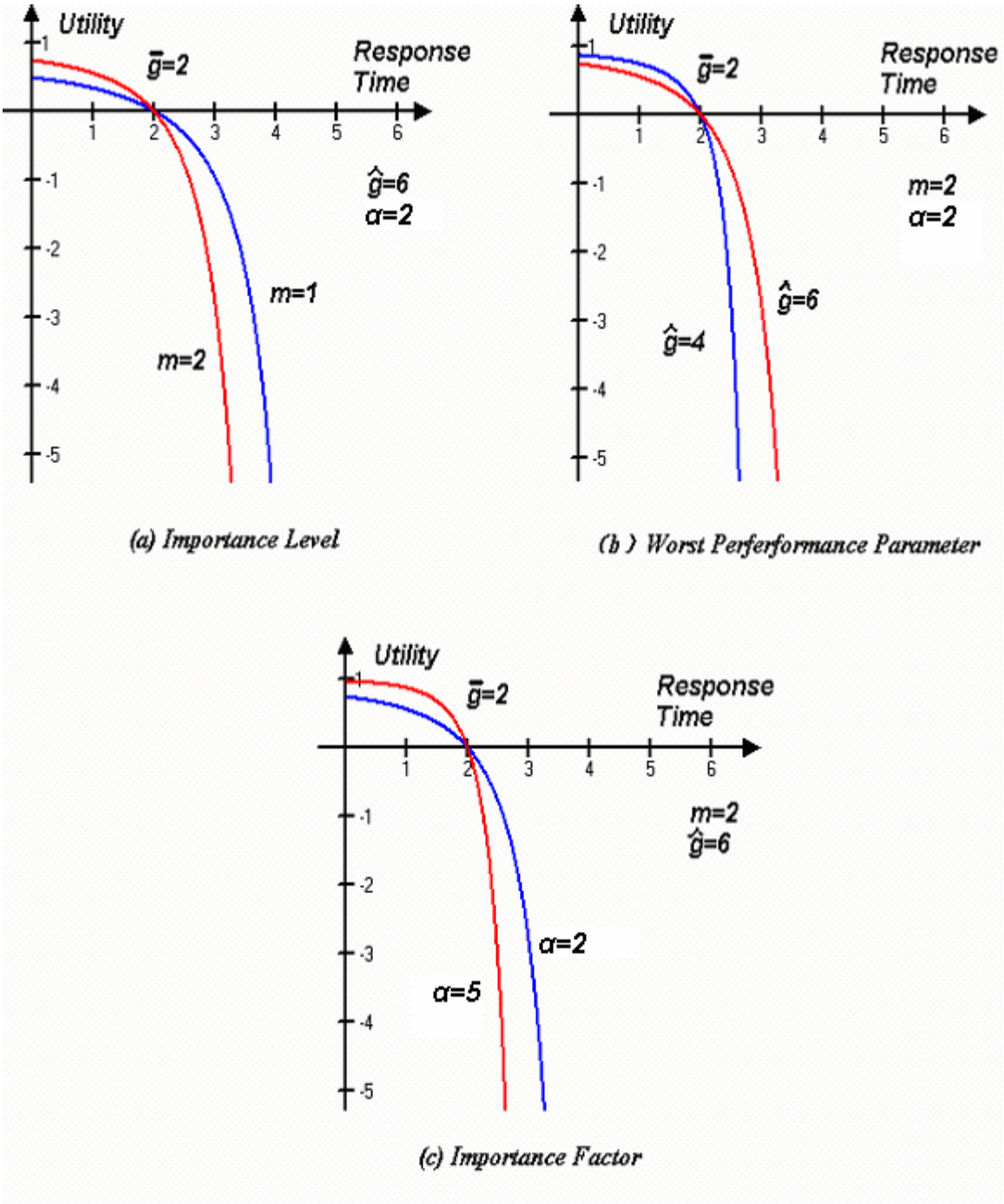

Figure 6 Utility functions for response time goals

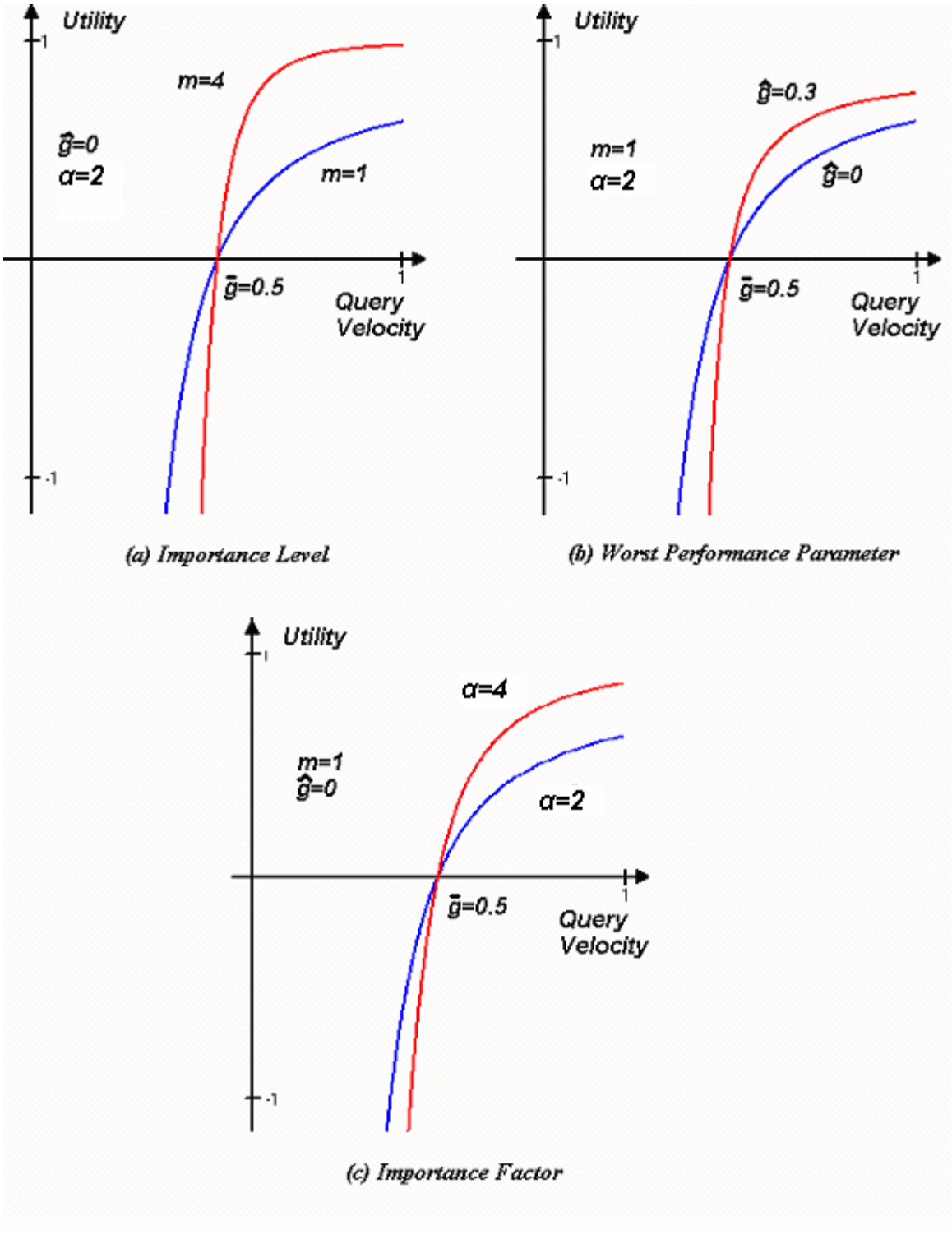

Figure 7 Utility functions for velocity goals

The parameters can control the shape of the curve of the utility function. The worst performance parameter $\hat{g}$ defines the upper bound of the domain of the utility function for response time goals, and the lower bound for query velocity goals. As shown in Figures 6 (b), for response time goals, the curve with a lower worst performance ($\hat{g} = 4$) is steeper than the curve with a higher worst performance ($\hat{g} = 6$). As shown in Figure 7 (b), for query velocity goals, the curve with a higher worst performance ($\hat{g} = 0.3$) is steeper than the curve with a lower worst performance ($\hat{g} = 0$). Both a lower worst performance for response time goals and a higher worst performance for query velocity goals, mean a narrower domain of the utility function, which results in a steeper curve of the utility function.

The importance factor parameter α influences the shape of the utility function in the same way as the importance level. As show in Figures 6 (c) and 7 (c), the higher the importance factor the steeper the utility function. The role of the importance factor is to provide systems with a means to fine tune the importance levels defined by users to reflect a proper hierarchy of workload importance.

4.3 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. The information used in the Classifier includes SLOs from

users, and query identification information and estimated query cost from the monitor component, which, in turn, comes from the DB2 QP (See Figure 5).

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 execution information from the monitor component is used to determine the performance of a query during runtime. The total cost of all queries in a service class is the resource demand of the service class, which is necessary to predict the performance of the service class and make resource control plans.

The workload classification method described above is a simple way to characterize the workload. We use it in Query Scheduler to show the feasibility of the general framework. There are other ways to classify queries. For example, queries can be classified based on query cost and then grouped into several groups according to size so that the queries can be manipulated based on their size. A two-level classification based on both performance goals and query cost can better characterize the workload by taking

both SLOs and query size into consideration, which allows more control over the workload. These alternatives are not the focus of this thesis and are left for future work.

4.4 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.

We denote the following symbols for a service class:

l: The number of terminal users in the service class.

C^i : The class cost limit of the service class at the i^{th} control interval.

V^i : The query velocity of the service class at the i^{th} control interval.

T^i : The average response time of queries in the service class at the i^{th} control interval.

w^i : The average waiting time of queries in the service class at the i^{th} control interval.

For a service class, the system can be viewed as a closed queuing model [LZGS84] with a single service center. There are l terminal users submitting queries interactively with zero think time to the service center, whose capacity is proportional to

the cost limit assigned to it. We are assuming that the system is always busy. Since the system total cost limit is fixed, the average service time of a query is fixed and we denote it as τ. Let us consider a standard service center with a cost limit equal to the average cost of a query, *C*. For this standard service center, there is only one query is served at a time. As per mean value analysis in queuing theory, the average queue length of the service center is $l-1$. The average response time T and wait time W are:

$$T = l\tau$$

$$W = (l-1)\tau$$

Now let us consider the performance of the service class at $(k-1)^{st}$ control interval with the class cost limit C^{k-1}. A service center assigned to the cost limit C^{k-1} with l terminal users is equivalent to C^{k-1}/C standard service centers each with $\left(C/C^{k-1}\right)l$ terminal users. The average response time T^{k-1} and wait time W^{k-1} at the $(k-1)^{st}$ control interval are:

$$T^{k-1} = \left(C/C^{k-1}\right)l\tau \qquad (4.1)$$

$$W^{k-1} = \left(\left(C/C^{k-1}\right)l-1\right)\tau$$

We have:

$$V^{k-1} = 1-W^{k-1}/T^{k-1} = C^{k-1}/\left(Cl\right)$$

Similarly, at the k^{th} control interval with the class cost limit C^k, the average response time T^k and wait time W^k are:

$$T^k = \left(C\ /C^k\right) l\tau \tag{4.2}$$

$$W^k = \left(\left(C\ /C^k\right) l - 1\right)\tau$$

We have:

$$V^k = 1 - W^k / T^k = C^k / \left(Cl\right) = V^{k-1} C^k / C^{k-1}$$

Because query velocity is less than *1*, we have the performance prediction model for query velocity as follows:

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

From the discussion about performance modeling for query velocity, we can easily derive the performance model for response time. Suppose the performance goals are changed from velocity goals to average response time goals. From Equations (4.1) and (4.2) we have the performance prediction model for average response time as follows:

$$\begin{aligned} T^k &= (C\ /C^k) l\tau \\ &= \left(C^{k-1} / C^k\right) T^{k-1} \end{aligned}$$

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

As shown in Figure 5, the performance modeling component is the Performance Solver in our implementation. It receives a new scheduling plan and 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.

4.5 System Monitor

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 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 the agent. 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.

4.6 Workload Control

The workload control component finds an optimal workload control 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.

4.6.1 Performance Optimization

Finding an optimal workload control plan can be described as the following optimization problem. We denote:

m_i : The importance level of service class *i*.

$\bar{g}_i$: The performance goal of service class *i*.

$\hat{g}_i$: The worst performance allowed for service class *i*.

g_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 the objective function in Section 4.2.3, the utility function in the Section 4.2.4, and the performance models in the Section 4.4, we have:

Objective function: $f = \sum u_i^k$ (4.3)

Utility function: $u_i^k = 1 - e^{\alpha m_i \frac{\bar{g}_i - g_i^k}{g_i^k - \hat{g}_i}}$ (4.4)

Performance model: $g_i^k = \begin{cases} g_i^{k-1} C_i^k / C_i^{k-1} & \text{Velocity goals} \\ g_i^{k-1} C_i^{k-1} / C_i^k & \text{Response time gaols} \end{cases}$ (4.5)

Replacing g_i^k in (4.4) with (4.5) and u_i^k in (4.3) with (4.4), the objective function becomes the function of the new workload control plan $(C_1^k, C_2^k, ..., C_n^k)$:

$$f = \sum u_i^k = f(C_1^k, C_2^k, ..., C_n^k)$$

with the constraint:

$$C_1^k + C_2^k + ... + C_n^k = C .$$

If the objective function has a global optimal value (maximum), we can find an optimal workload control plan for the next control interval. Since the utility function u_i^k only has a single variable C_i^k after replacing g_i^k in (4.4) with (4.5), we have the second derivative for u_i^k as follow:

$$\frac{\partial^2 u_i^k}{\partial (g_i^k)^2} = -e^{\alpha m_i \frac{\bar{g}_i - g_i^k}{g_i^k - \hat{g}_i}} \left(\alpha m_i \frac{\bar{g}_i - \hat{g}_i}{\left(g_i^k - \hat{g}_i \right)^2} \right)^2 - 2e^{\alpha m_i \frac{\bar{g}_i - g_i^k}{g_i^k - \hat{g}_i}} \left(\alpha m_i \frac{\bar{g}_i - \hat{g}_i}{\left(g_i^k - \hat{g}_i \right)^3} \right)$$

Both g_i^k and $\bar{g}_i$ are not less than $\hat{g}_i$ for a velocity goal and larger than $\hat{g}_i$ for a response time goal, so the second derivative for u_i^k is less than zero. u_i^k is therefore a concave function [Binm83]. As the sum of the utility functions, the objective function is also a concave function [Avri03]. It therefore has a global maximum, which corresponds to the optimal workload control plan.

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.

4.6.2 Solving Techniques

As mentioned earlier, the problem to manage multiple service classes towards their SLOs can be solved by maximizing the objective function:

$$f(C_1^k, C_2^k, ..., C_n^k), \text{ where } C_1^k + C_2^k + ... + C_n^k = C.$$

There exists a large body of research work for solving optimization problems. For concave functions, Ellipsoid methods, sub-gradient methods, cutting-plane methods, interior-point methods, etc [Avri03] [BV04] can be used to solve the optimization problems efficiently with the global optimum. If the objective function is not a concave function, there are other methods to solve the optimization problems, such as hill climbing, simulated annealing, quantum annealing, Tabu search, beam search, genetic algorithms, ant colony optimization, evolution strategy, stochastic tunneling, differential evolution, particle swarm optimization, harmony search, bees algorithm, etc [Avri03].

The drawbacks of these algorithms are that they are inaccurate and may prematurely halt at a local optimum [MB03].

Since the utility functions we choose have a second derivative, and the objective function has all second partial derivatives, we choose to solve it using quasi-Newton methods. The first step to solve the problem by using numerical methods is to apply the Lagrange method to produce a set of nonlinear equations [FB93].

$$F = f(C_1^k, C_2^k, ..., C_n^k) + \lambda(C_1^k + C_2^k + ... + C_n^k - C)^{'}$$

where λ is the Lagrange multiplier. Setting the first partial derivatives of F to zero, we have

$$\begin{cases} g_1 = \dfrac{\partial F}{\partial C_1^k} = 0 \\ g_2 = \dfrac{\partial F}{\partial C_2^k} = 0 \\ \cdots \\ g_n = \dfrac{\partial F}{\partial C_n^k} = 0 \\ g_{n+1} = \dfrac{\partial F}{\partial \lambda} = 0 \end{cases}$$

We denote the set of equations as:

$$\mathbf{G}(\mathbf{x}) = 0$$

where $\mathbf{G} = (g_1, g_2, \cdots, g_{n+1})^t$, $\mathbf{x} = (x_1, x_2, \cdots, x_{n+1})^t = (C_1^k, C_2^k, \cdots C_n^k, \lambda)^t$

The classic method to solve the nonlinear equations $\mathbf{G}(\mathbf{x}) = 0$ is Newton's method [FB93]. However it is expensive with $O(n^3)$ computational complexity. There are many quasi-Newton methods with reduced complexity [MC79]. Broyden's method with $O(n^2)$ computational complexity is one of the methods that can be implemented easily and efficiently. The algorithm is shown in Figure 8.

Input: initial approximation $\mathbf{x}$; tolerance *Tol* and maximum number of iterations M
Output: solution $\mathbf{x}$ or a message that the number of iterations was reached

1 $\mathbf{A_0} = \mathbf{J}(\mathbf{x})$, $\mathbf{v}=\mathbf{F}(\mathbf{x})$, where $\mathbf{J}(\mathbf{x})_{i,j} = \dfrac{\partial g_i(\mathbf{x})}{\partial x_j}$
for $1 \le i, j \le n+1$.
2 $\mathbf{A} = \mathbf{A}_0^{-1}$.
3 $k = 1$; $\mathbf{s} = -\mathbf{A}\mathbf{v}$; $\mathbf{x} = \mathbf{x} + \mathbf{s}$.
4 while ($k \le M$) do 5 - 13.
5 $\mathbf{w} = \mathbf{v}$; $\mathbf{v} = \mathbf{F}(\mathbf{x})$; $\mathbf{y} = \mathbf{v} - \mathbf{w}$.
6 $\mathbf{z} = -\mathbf{A}\mathbf{y}$.
7 $p = -\mathbf{s}^t\mathbf{z}$.
8 $\mathbf{C} = p\mathbf{I} + (\mathbf{s} + \mathbf{z})\mathbf{s}^t$.
9 $\mathbf{A} = (1/p)\mathbf{C}\mathbf{A}$.
10 $\mathbf{s} = -\mathbf{A}\mathbf{v}$.
11 $\mathbf{x} = \mathbf{x} + \mathbf{s}$.
12 if $\|\mathbf{s}\| < TOL$ output ($\mathbf{x}$); goto 15.
13 $k = k + 1$.
14 output("Maximum number of iterations reached").
15 end.

Figure 8 Broyden algorithm

4.6.3 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 algorithm shown in Figure 9 to execute the scheduling plan and perform admission control:

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

Figure 9 Admission control algorithm

4.7 Summary

Query Scheduler is a proof-of-concept implementation of the framework for workload adaptation. It is implemented outside DB2 and uses its workload manager DB2 QP. In the implementation, multiple SLOs are encapsulated into an objective function, which is the sum of the utilities of the service classes. The utility of a service class measures how well the system is meeting the SLO of the service class, and can be described by a function of

the performance goal and the workload importance of the service class. A general form of utility functions is proposed. Query Scheduler uses a queuing network model to predict the performance. A workload control plan, a set of class cost limits, is derived from maximizing the objective function, which is solved by numerical methods.

Chapter 5

Evaluation of Query Scheduler

In this Chapter 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 with DB2 QP, which is typical of the level of control available in current DBMSs.

Because the Query Scheduler is currently implemented outside DB2, the overhead associated with managing queries with the Query Scheduler means that it is impractical to try to manage online transaction processing (OLTP) workloads (see Section 5.2.1), which are composed mainly of small queries. We first focus on management of large queries, such as OLAP queries found in decision support systems (DSSs), to evaluate the effectiveness of Query Scheduler in providing differentiated service to workload classes with different SLOs. We then try to adapt to mixed workloads with both OLTP and OLAP queries. The OLTP queries are managed indirectly through controlling OLAP queries to affect the performance of OLTP queries.

5.1 Evaluation with OLAP Workload

We designed a set of experiments to show the effectiveness of Query Scheduler relative to that of DB2 QP. Section 5.1.1 introduces the OLAP workloads used for the experiments and the information about the experimental environment. Section 5.1.2 describes the approach to determining the total cost threshold for the experiments. The set of experiments are presented in the Section 5.1.3, and Section 5.1.4 analyzes the experiment results.

5.1.1 Workload

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 Version 8.2 and Query Patroller as supporting components.

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 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 for each class. The intensity is varied overtime in each experiment as shown in Figure 10. 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

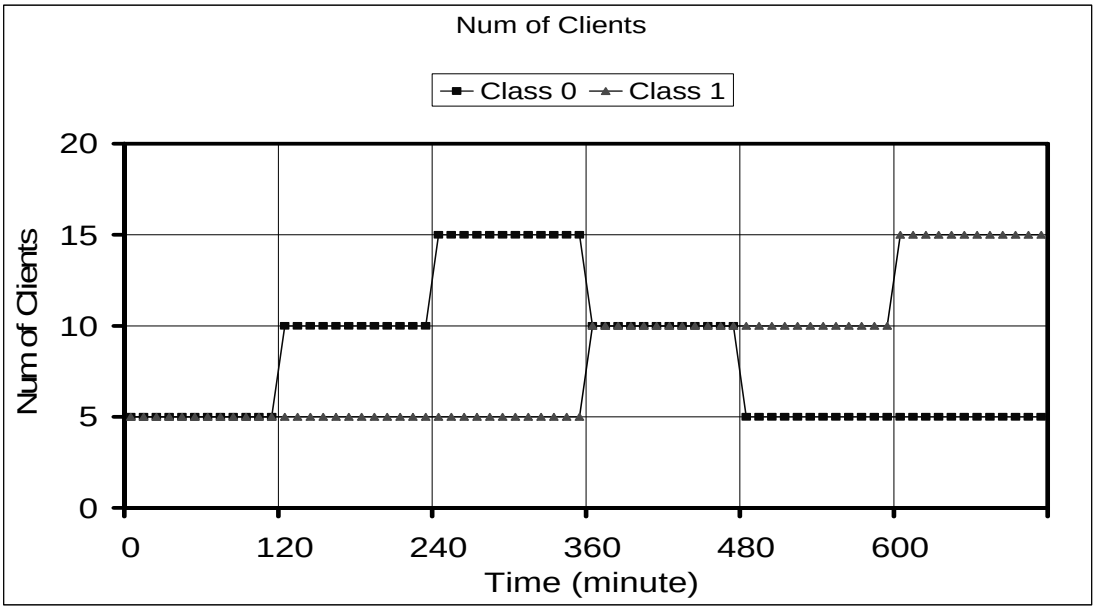

Figure 10 Workload intensity

3 where 15 clients from Class 0 and 5 clients from Class 1 are issuing queries simultaneously.

5.1.2 Determine the Total 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 11) and total cost vs. throughput (Figure 12) are plotted to determine the total cost limit that keeps the system saturated. We find that the circled point in Figure 11 and 12 with total cost limit of 300,000 timerons is 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.

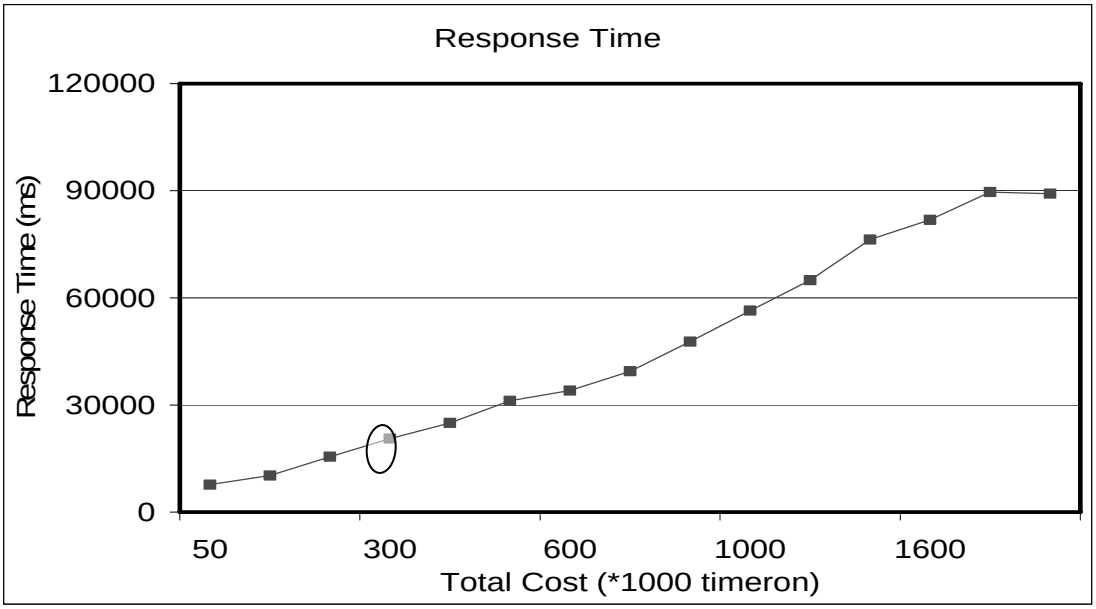

Figure 11 Response time vs. total cost

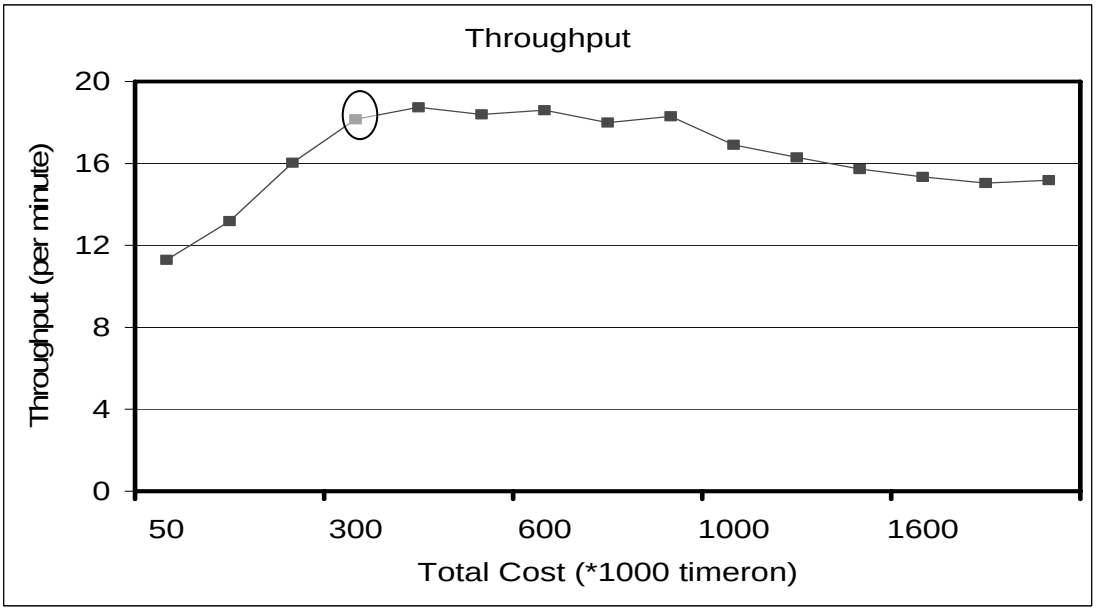

Figure 12 Throughput vs. total cost

5.1.3 Experiments

The following set of experiments shows the effectiveness of Query Scheduler relative to DB2 QP. All the experiments use the workload described in Section 5.1.1 with total cost limit determined in Section 5.1.2. The workload is generated by a workload generator and Query scheduler performs workload control every 10 minutes.

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 13

Class Control with DB2 QP

In this experiment, we use DB2 QP as the performance controller. The typical way to use DB2 QP is to partition workload into three groups based on the cost of queries: large, medium and small and restrict the number of queries from the large and the medium groups. As suggested by IBM researchers, the cost threshold for the large query group is chosen as the lowest percentile cost of 95% of all queries and 80% for medium group:

Large: $\text{cost} > \textit{the lowest percentile cost of } 95\%$

Medium: $\textit{the lowest percentile cost of } 80\% < \text{cost} \leq \textit{the lowest percentile cost of } 95\%$

Small: $\text{cost} \leq \textit{the lowest percentile cost of } 80\%$

The number of queries from the large and the medium groups are set as 2 and 5 respectively for the workload we use.

DB2 QP can also perform priority control over the workload to provide differentiated services. A service class can be assigned priority. The queries in a service class with higher priority are always admitted before the queries in a service class with lower priority. 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 14. We then turn off priority control. The result is shown in Figure 15.

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, which are properly selected through experiments. 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 16 for the query velocity and in Figure 17 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 18 for the query velocity and in Figure 19 for the adjustment of class cost limits.

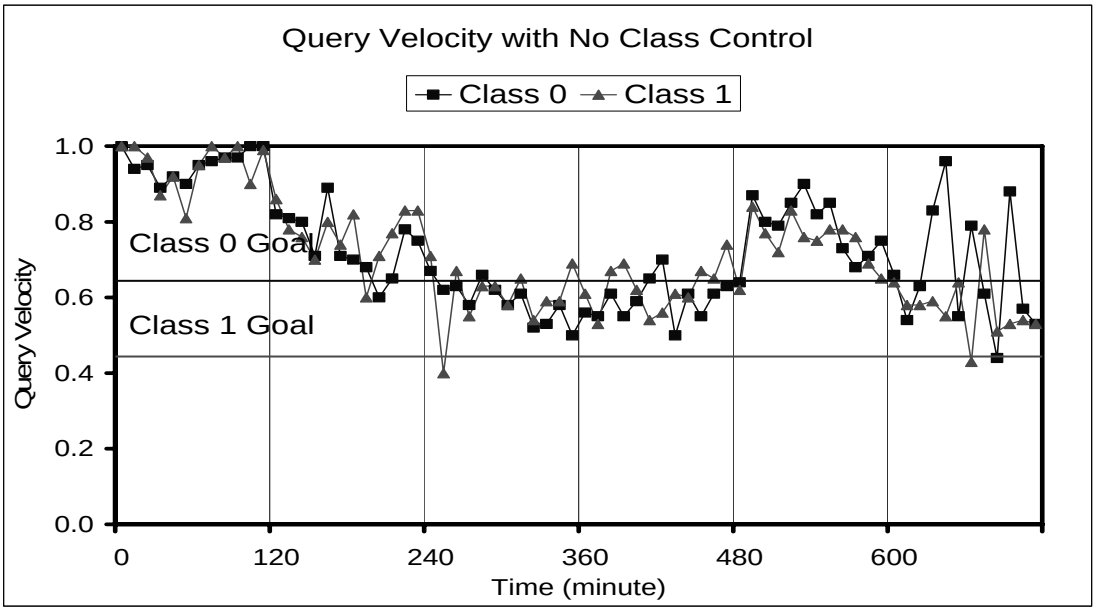

Figure 13 Query velocity with no control

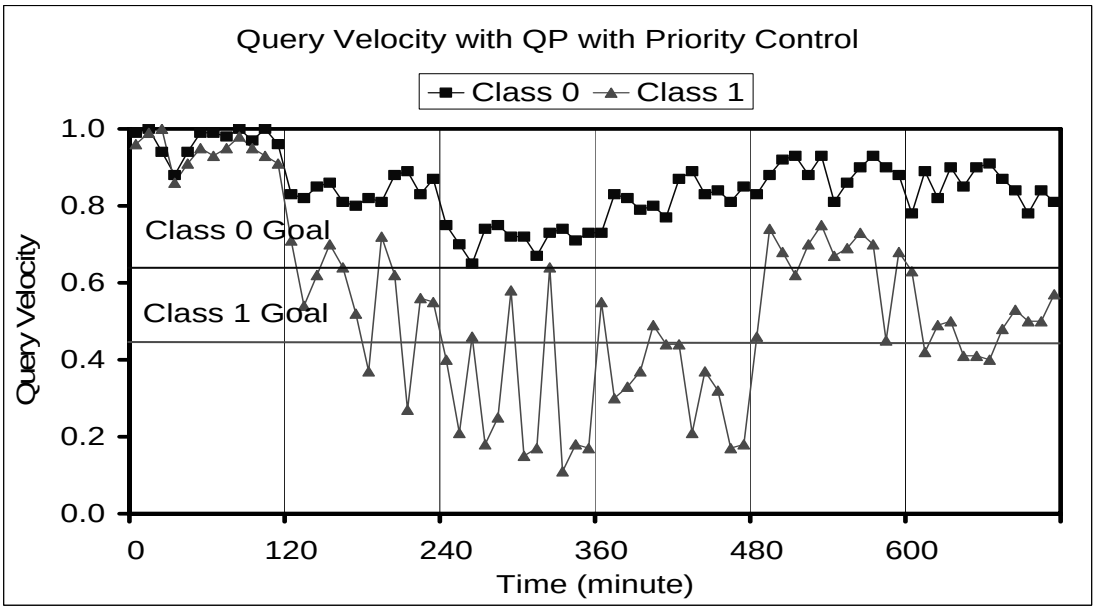

Figure 14 Query velocity with DB2 QP with priority control

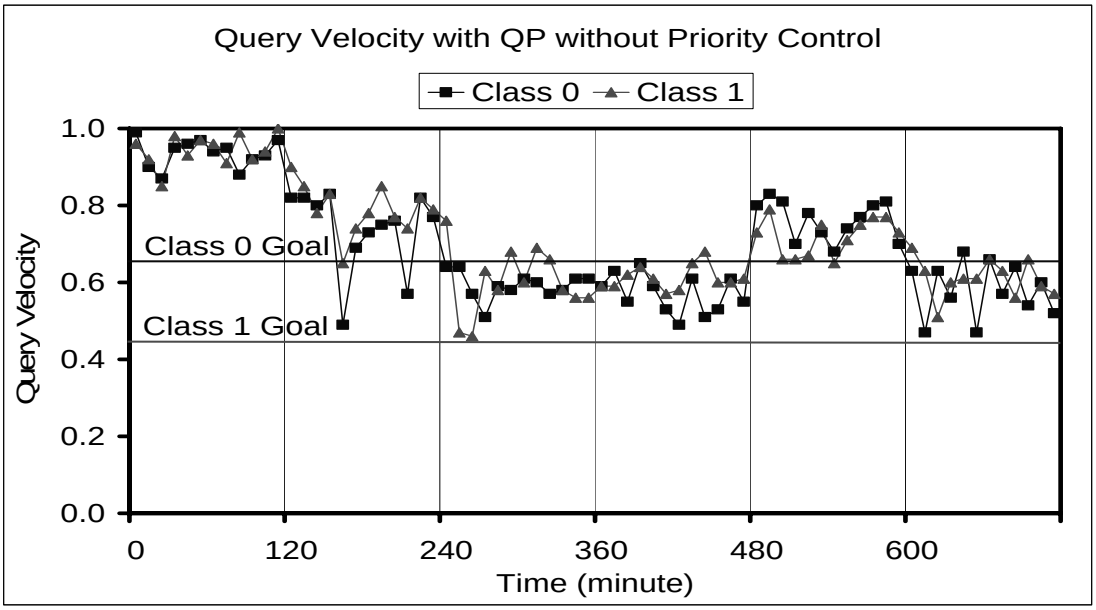

Figure 15 Query velocity with DB2 QP without priority control

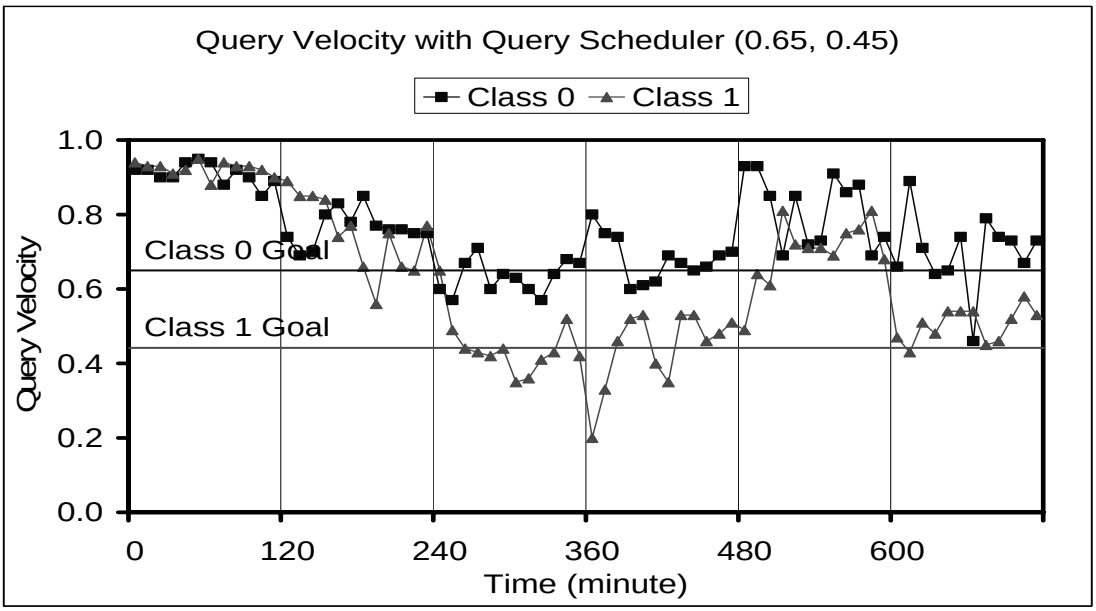

Figure 16 Query velocity with Query Scheduler with goals (0.65, 0.45)

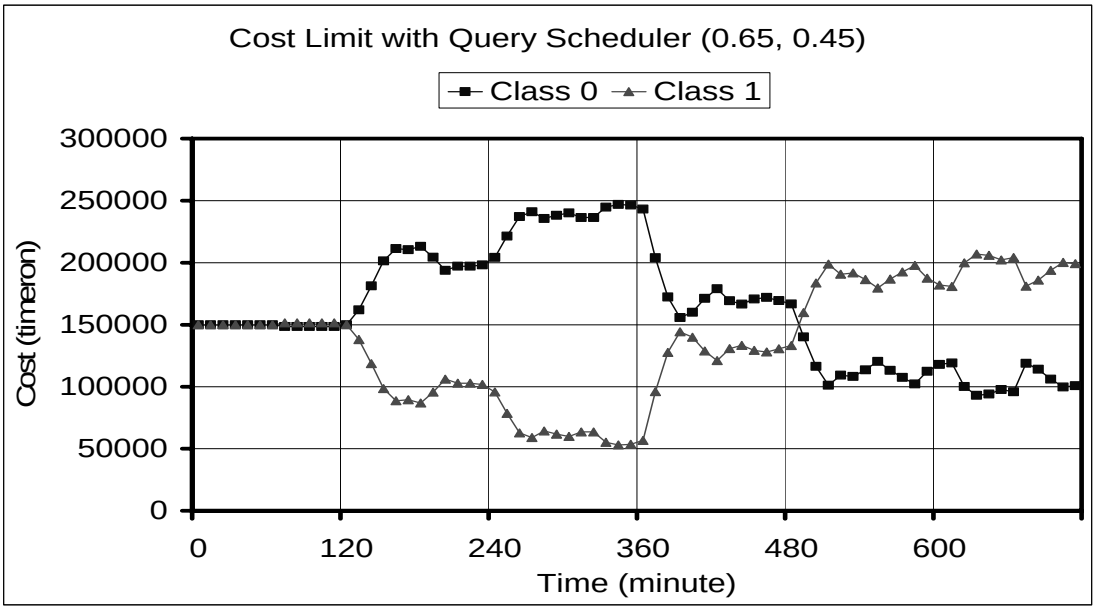

Figure 17 Adjustment of class cost limits with Query Scheduler with goals (0.65, 0.45)

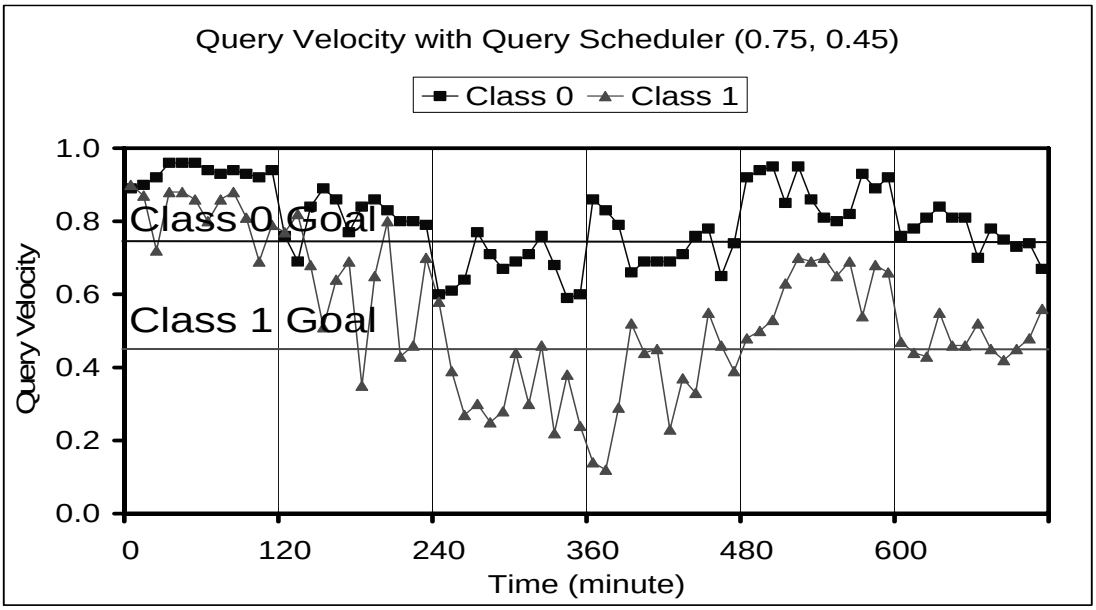

Figure 18 Query velocity with Query Scheduler with goals (0.75, 0.45)

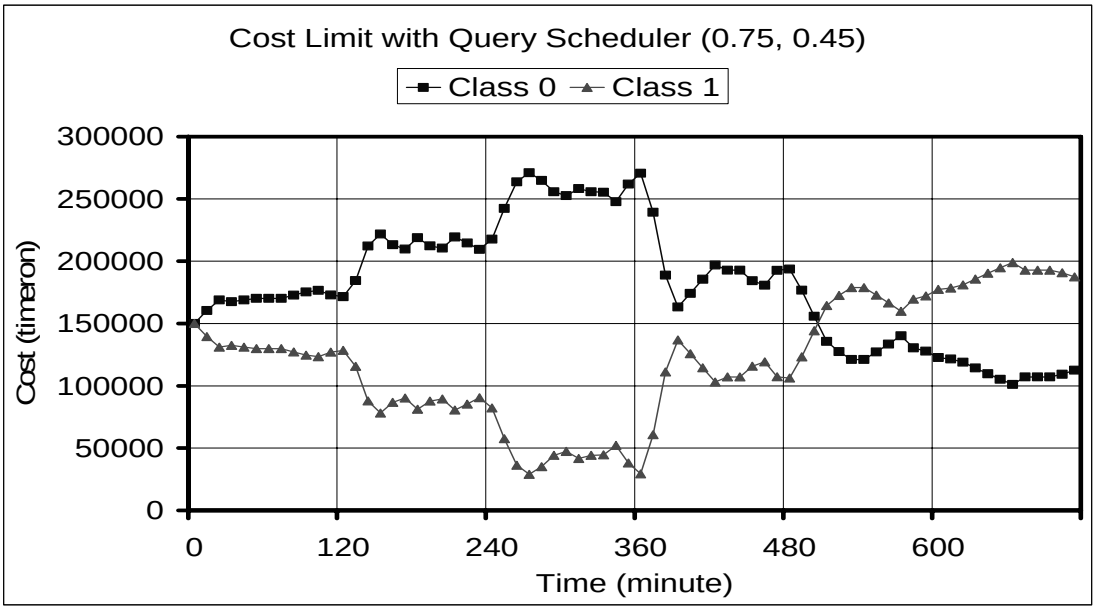

Figure 19 Adjustment of class cost limits with Query Scheduler with goals (0.75, 0.45)

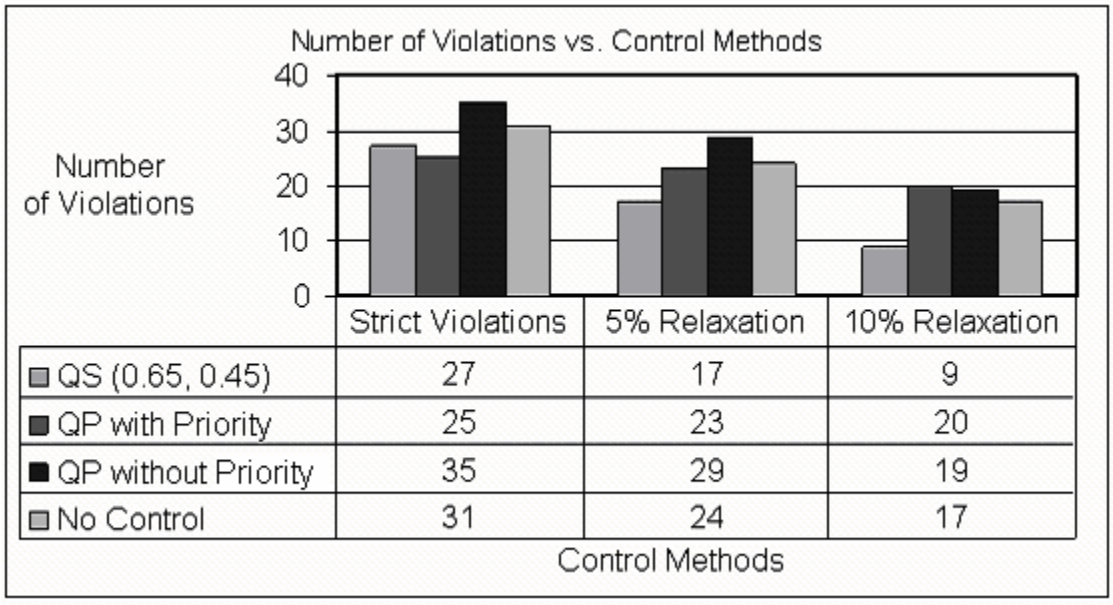

Figure 20 Comparison of number of violations of control methods

Table 2 Average query velocity in each period

Period	QS(0.65,0.45)		QP		QPNP		NoControl	
	Class 0	Class 1	Class 0	Class 1	Class 0	Class 1	Class 0	Class 1
1	0.91	0.92	0.97	0.94	0.94	0.94	0.95	0.95
2	0.77	0.75	0.84	0.57	0.74	0.79	0.74	0.77
3	0.63	0.45	0.72	0.29	0.59	0.60	0.59	0.60
4	0.68	0.44	0.82	0.35	0.57	0.61	0.59	0.62
5	0.82	0.70	0.89	0.66	0.76	0.72	0.78	0.75
6	0.70	0.50	0.85	0.49	0.58	0.61	0.67	0.58

5.1.4 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 14, 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 15, the query velocity curves of both classes are similar to the case of No class control (Figure 13). As shown in Figure 16 and 18 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 assigns 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 14. Query Scheduler dynamically adjusts the class cost limits based on the performance as shown in Figures 17 and 19. 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 16 and 18. 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 14, while Query Scheduler is able to keep both classes converging on their performance goals as shown in Figure 16 and 18.

Statistics also support the claim that Query Scheduler can provide better quality of differentiated service than the other control methods. Figure 20 shows the number of SLO violations for the four control methods. When the performance goals are set as 0.65 and 0.45 for Class 0 and Class 1 respectively, Query Scheduler results in 27 SLO violations. The number is only 2 more than DB2 QP with priority control, and better than DB2 QP without priority control and No Control. If we relax the performance goals 5 percent (0.4275 for Class 0 and 0.6175 for Class 1) and 10 percent (0.405 for Class 0 and 0.585 for Class 1), Query Scheduler is much better than the other three control methods. This is to say Query Scheduler can keep the performance of the two classes in a narrower band around the performance goals than other control methods do.

We further calculate the average query velocity for each period for all the control methods shown in Table 2. The real performance with Query Scheduler is better than the performance goals in Periods 1, 2, 5 and 6, except for Class 1 with DB2 QP without priority control. When workload is heavy in Period 3 and 4, the real performance with Query Scheduler control is closely aligned with the performance goals. The biggest violation is only $(0.65-0.63)/0.65=3.1\%$ from the performance goal, compared with $(0.45-0.29)/0.45=35.6\%$ for DB2 QP with priority control, $(0.65-0.57)/0.65=12.3\%$ for DB2 QP without priority control, and $0.65-0.59=9.2\%$ for No Control.

In conclusion, Query Scheduler can anticipate the performance changes and control the performance of all service classes oscillating around their performance goals in a narrower band than the other control methods.

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 16). 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 18). 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 18. 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 17 and 19, 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 17 and 19. 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 14.

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

5.2 Evaluation with Mixed Workloads

Mixed workloads consisting of both OLAP and OLTP queries are common. Controlling mixed workloads with our framework is more complex than controlling a homogenous OLAP workload. In this section we discuss an approach to adapting mixed workloads with both OLAP and OLTP queries. First, we analyze the overhead imposed by the Query Scheduler in Section 5.2.1 and conclude that it is not effective to manage OLTP workloads directly. Second, we propose an indirect approach to managing OLTP

workloads through managing the concurrent OLAP workloads in Section 5.2.2. We then present the mixed workloads for the experiments in Section 5.2.3 and describe the set of experiments in Section 5.2.4. Finally the analysis of the experimental results is presented in Section 5.2.5.

5.2.1 The Overhead of Query Scheduler

The overhead of Query Scheduler consists of query interception, acquisition of query information, control logic and query release as shown in Figure 21. 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 [IBM03B] 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.

We measured the throughputs for OLAP and OLTP workloads with DB2 QP turned on and off. When DB2 QP is turned on, a pre-determined total cost limit is set to keep the system from overloading. The OLAP workload is composed of 22 TPC-H [TPC] queries and the OLTP workload consists of 5 types of transactions from TPC-C [TPC]

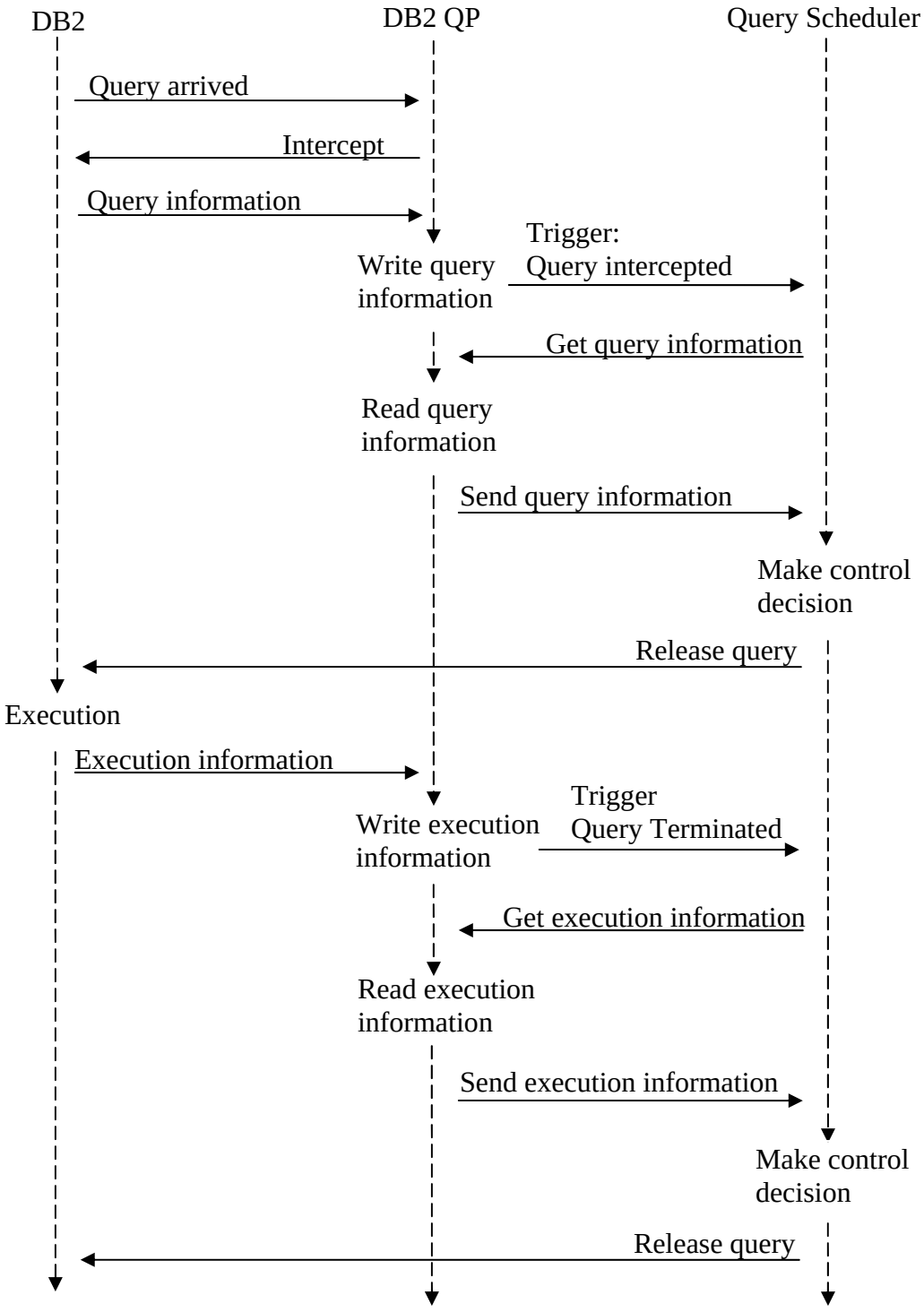

Figure 21 Overhead of Query Scheduler

benchmark. As shown in Figure 22, 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 Figure 23, 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 Figure 21

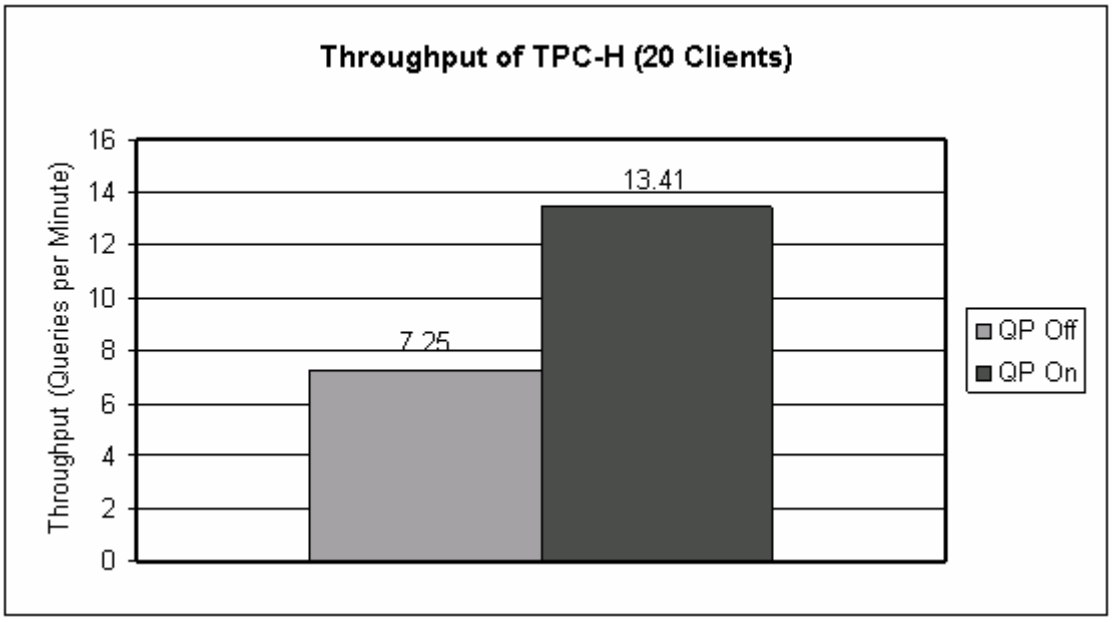

Figure 22 The effect of controlling OLAP workload with DB2 QP

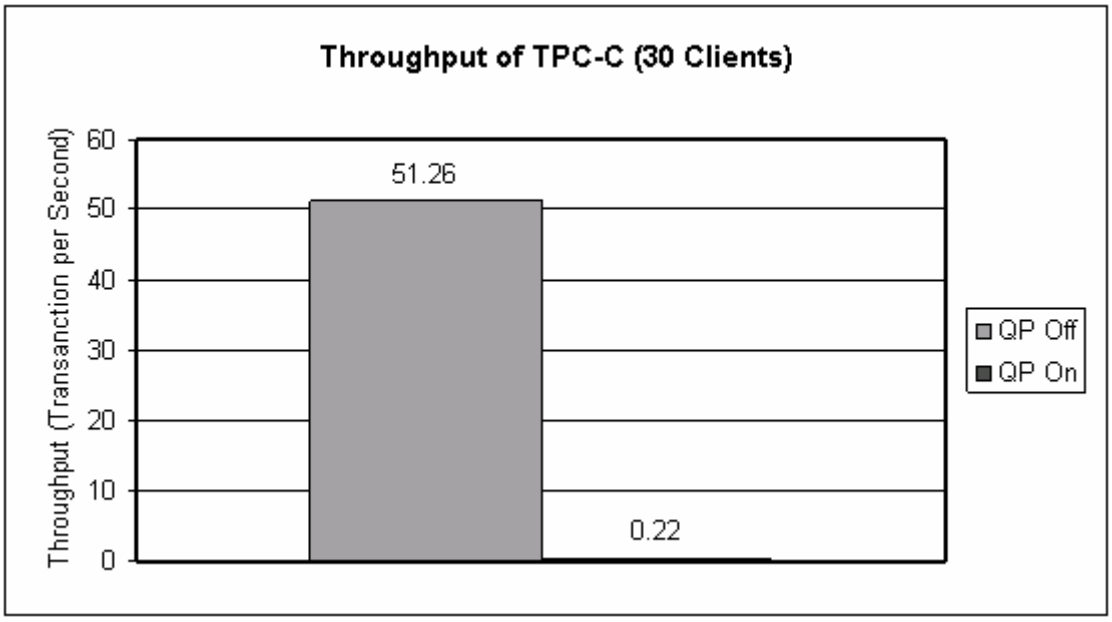

Figure 23 The effect of controlling OLTP workload with DB2 QP

5.2.2 Adapting OLTP Workload

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 workloads. 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 limits of competing OLAP classes and can decrease resources allocated to OLTP queries by raising the cost limits of competing OLAP classes. To do so, we configure DB2 QP to intercept OLAP queries and bypass OLTP queries.

Performance Modeling for the OLTP Class

The performance model for OLAP classes is the same as we described in Section 4.4. The performance model for the OLTP class, however, is different. First, the system does not control the OLTP class directly so there is no clear division between waiting time and execution time. Second, the total cost of OLTP queries at any one time is unknown. Third, OLAP queries tend to be I/O intensive whereas OLTP are CPU intensive.

We experimentally justified our decision to indirectly manage OLTP workloads by directly managing the OLAP workloads. We measured the average response time of the OLTP class relative to the sum of cost limits of all OLAP classes (Figure 24). 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.

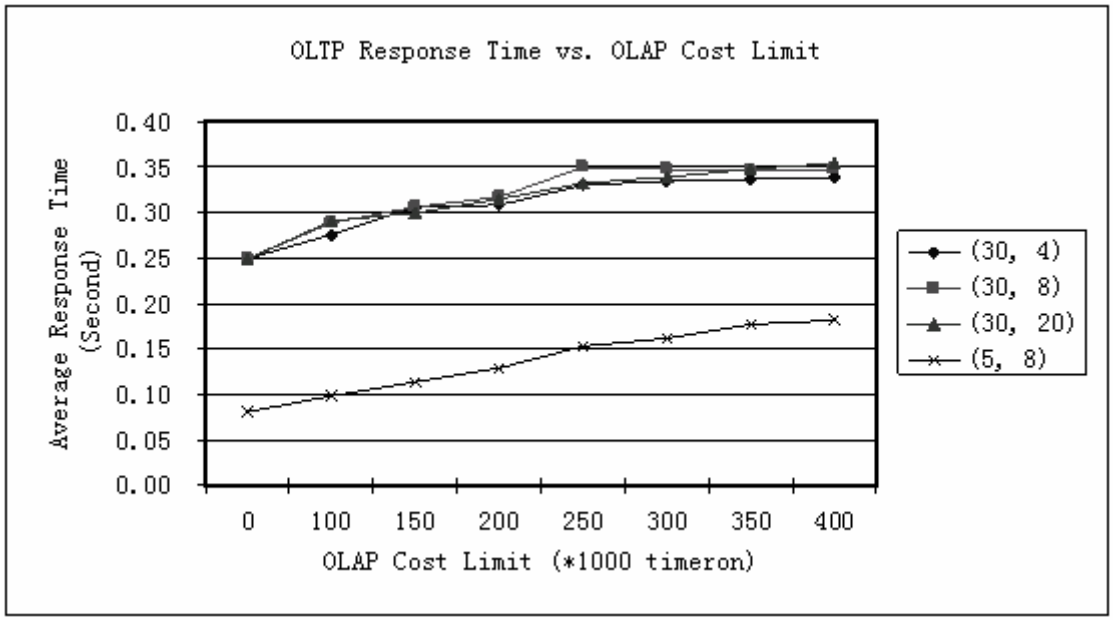

Figure 24 OLTP performance vs. OLAP cost limit

Monitoring OLTP Queries

It is necessary to monitor the performance of the OLTP class, for which we use an average response time metric, in order to make control decisions for the mixed workload. Since we turned off DB2 QP for the OLTP workload, we need other approaches for acquiring the information. The DB2 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 incurs too much overhead, nor too large, which decreases accuracy.

5.2.3 Workloads

The computer system used is the same as that described in Section 5.1.1. The TPC-H and TPC-C benchmarks [TPC] serve 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 with 30GB data. The tables for the two workload types are 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 consists of three workload classes; two classes of TPC-H queries and one class of TPC-C queries, all submitted by interactive clients. Each class has a performance goal. Each client submits queries one after another with zero think time. Workload intensity is controlled by the number of clients for each class (Figure 25). The number of clients for an OLAP class varies from 2 to 6 at any one time and the number of clients for the OLTP class varies from 15 to 25 at any one time. Each test run lasts 24 hours and is broken down into 18 80-minute periods. Workload intensity is consistent within a given period.

Class 1 and Class 2 are 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 has a higher query velocity goal, which means that its queries should be delayed less than queries of Class 1. Class 3 is the OLTP class with the highest

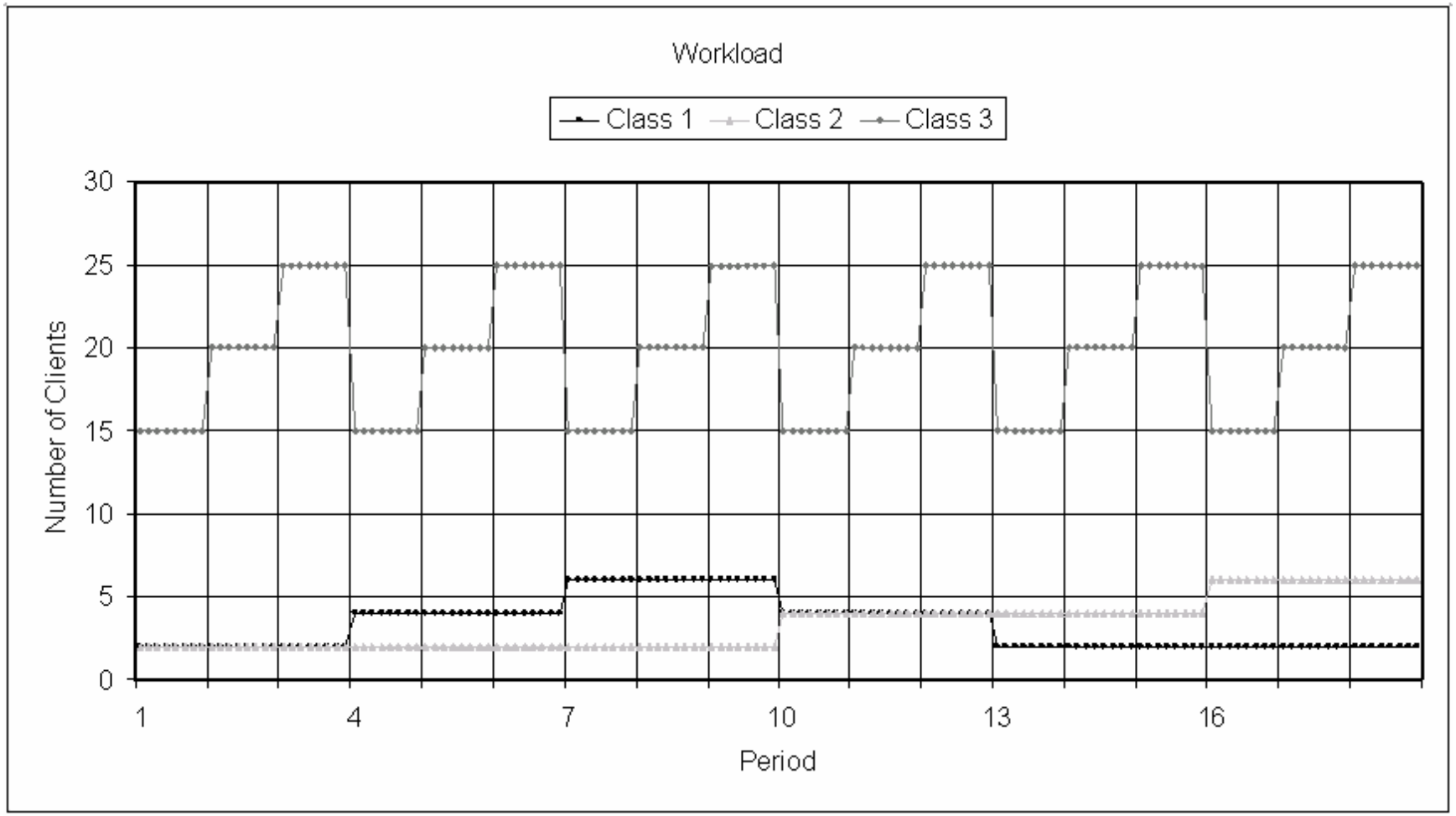

Figure 25 Workloads

importance level of 3, and is assigned an average response time goal 0.25 seconds as its performance goal. We determine experimentally that the goals for the three classes are reasonable. We see from Figure 25 that the heaviest workload is in period 18 where two clients from Class 1, six clients from Class 2 and twenty-five clients from Class 3 are issuing queries simultaneously.

5.2.4 Experiments

The following set of experiments shows 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 workloads shown in Figure 25 with total cost limit 300,000 timerons. The OLAP and OLTP workloads are generated using different workload generators and Query scheduler performs workload control every 10 minutes.

No Class Control

In this experiment, no control is 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 26.

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 run experiments with priority control turned on and off. For the case where priority control is turned on, we set the priority of Class 2 higher than that of Class 1. The results of this experiment are shown in Figure 27. For the case where priority control is turned off, we observe that the performance is similar to the case with no control so the results are not presented here.

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 28. Figure 29 shows the adjustment of class cost limits.

5.2.5 Analysis

Control over the 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 work

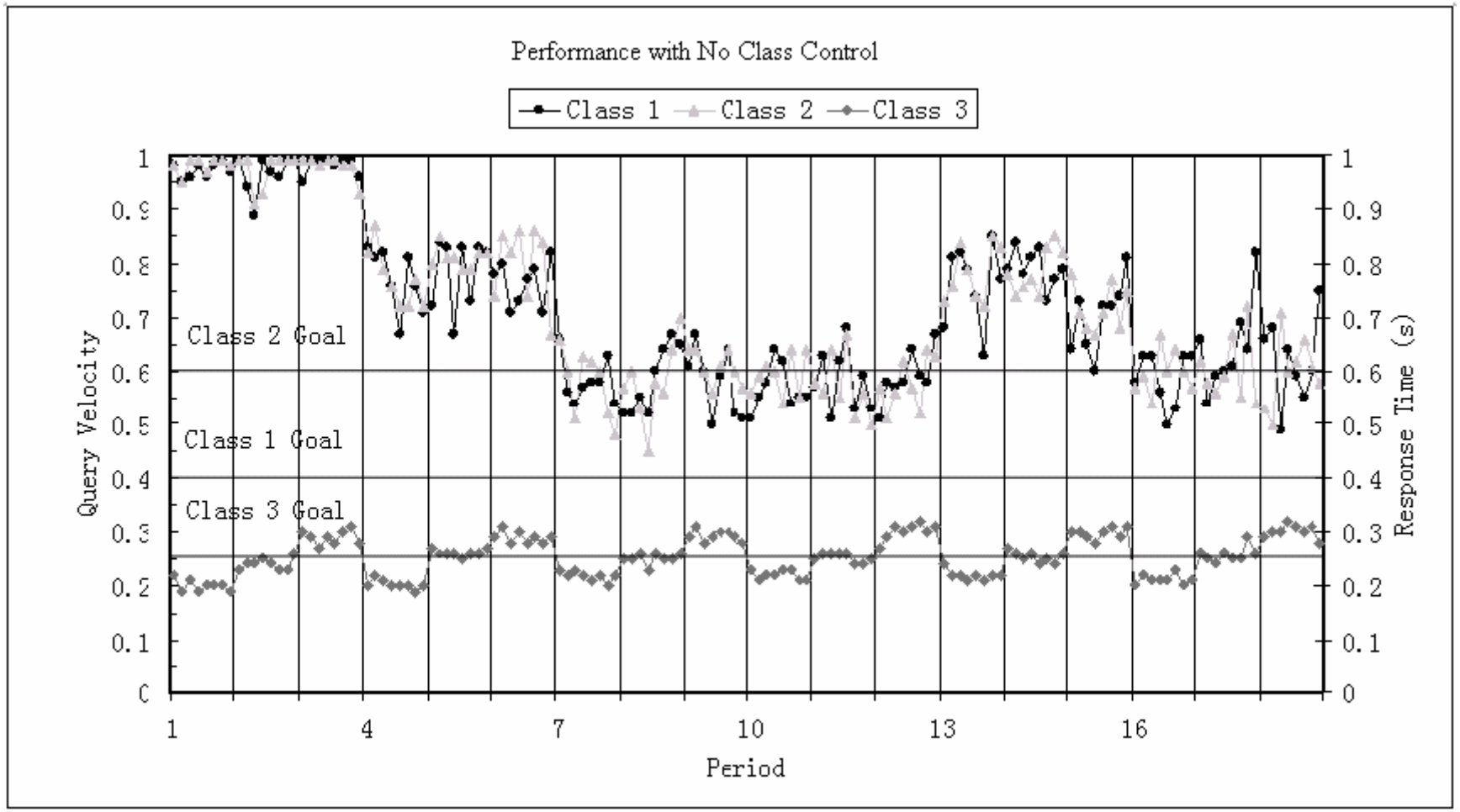

Figure 26 No class control

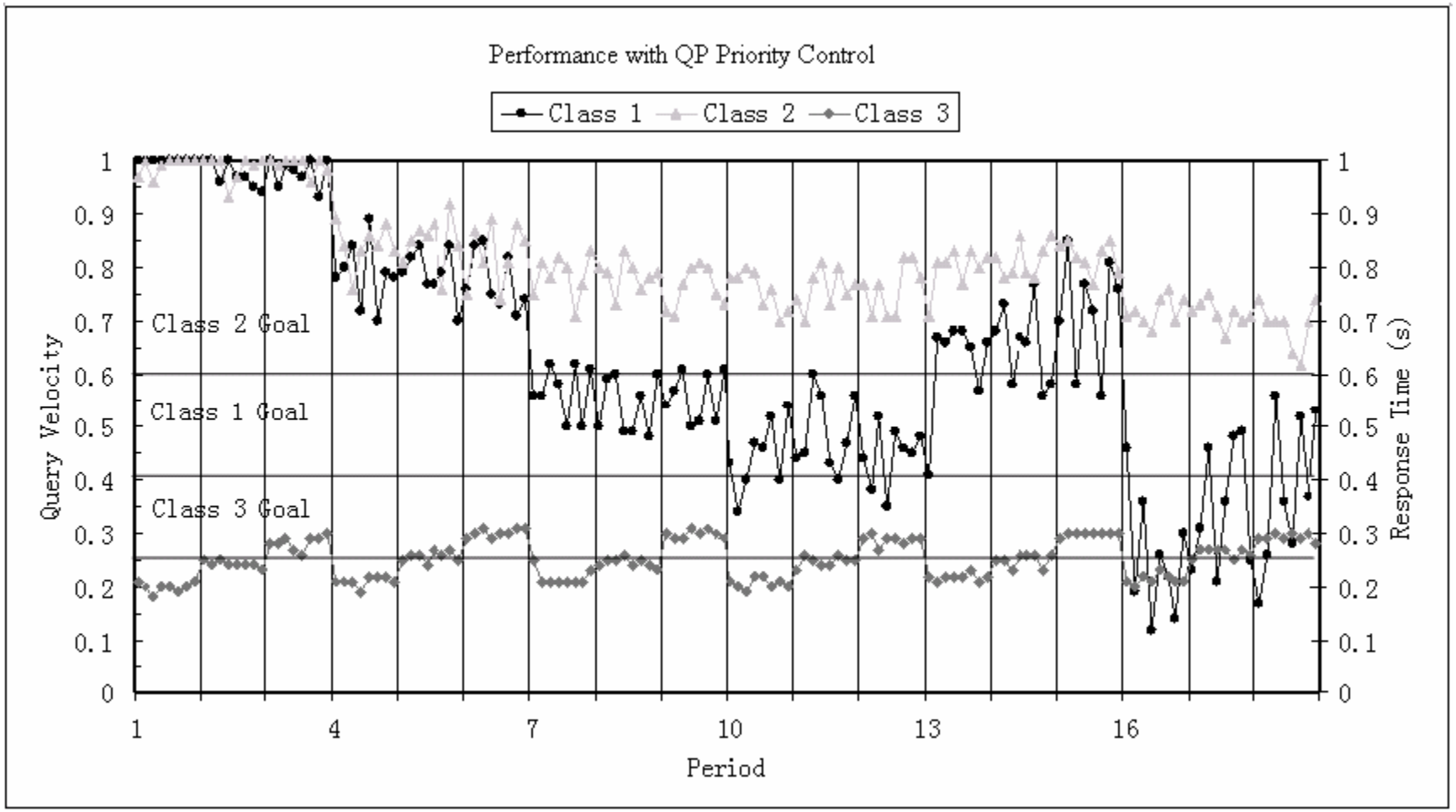

Figure 27 DB2 QP priority control

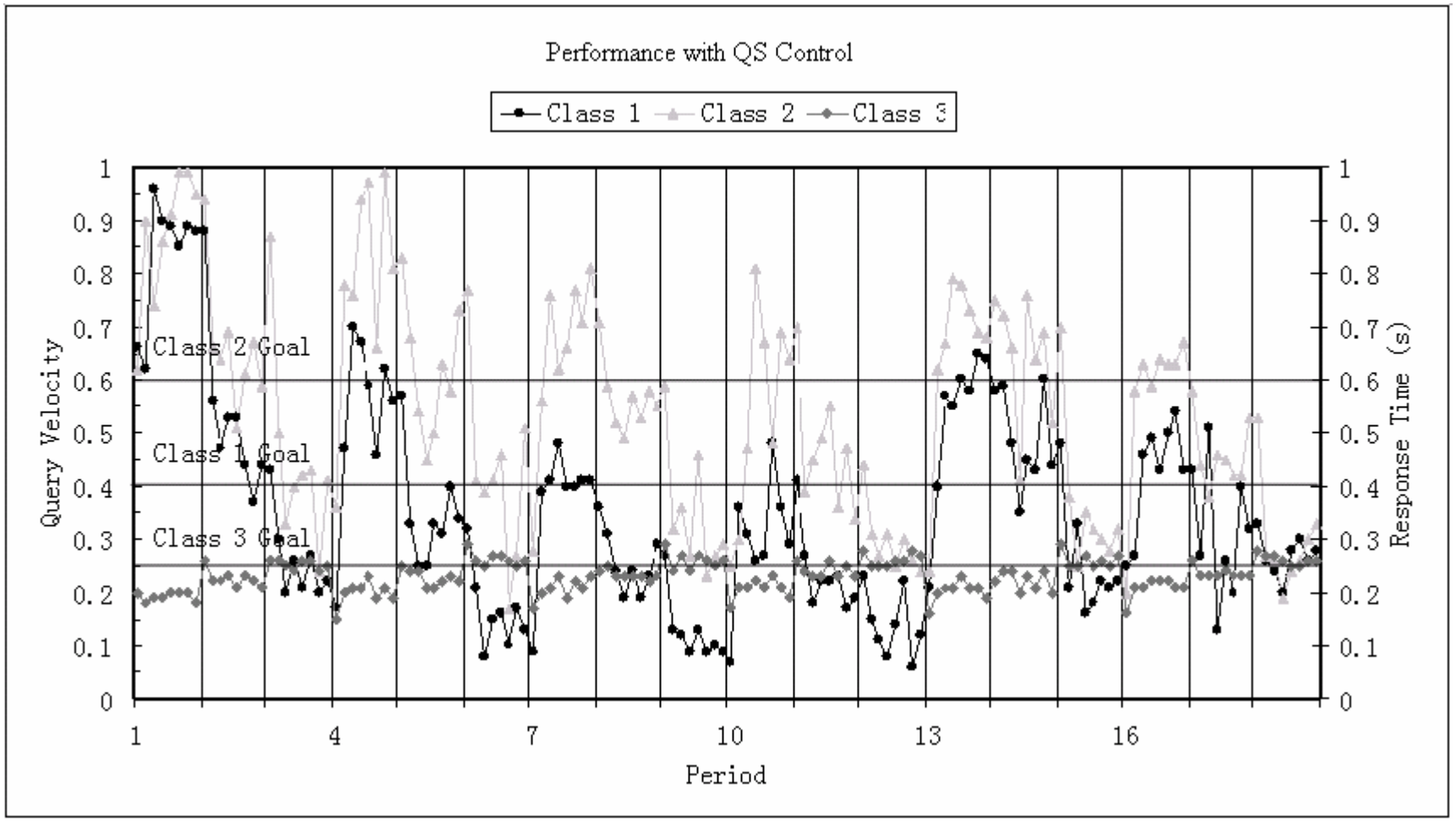

Figure 28 Query Scheduler control

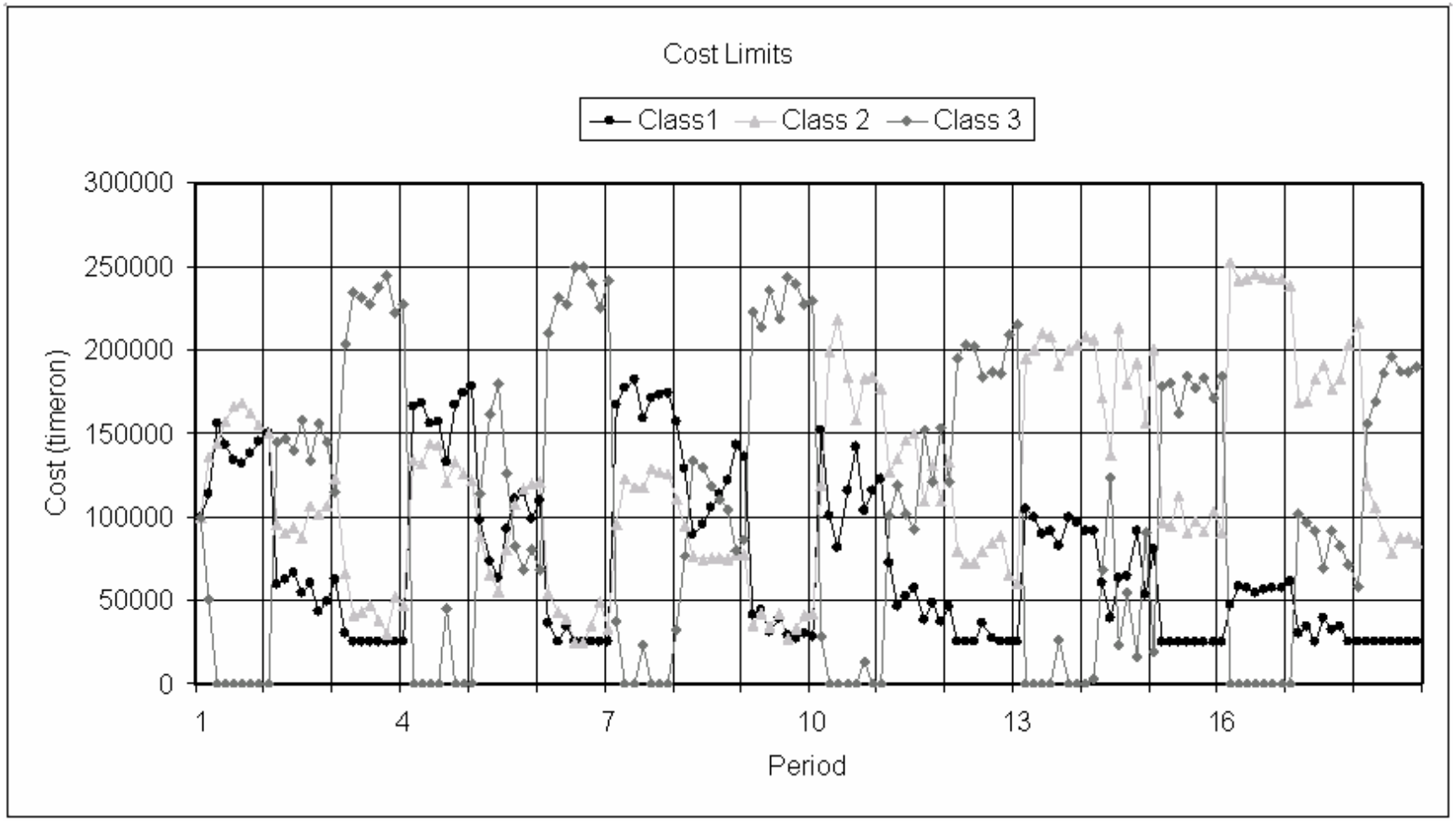

Figure 29 Adjustment of class cost limits with Query Scheduler control

-load 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 29) 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 27, Class 2 always performs better than Class 1. Figure 28 shows that Query Scheduler can provide differentiated service to both OLAP and OLTP classes. Class 3 meets its 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 in this case of mixed workloads. DB2 QP sets a static cost limit on the OLAP workload 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 27, 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 28, 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.

We summarized the number of SLO violations for all three services classes as shown in Table 3. The number of SLO violations for the OLTP workload with Query Scheduler control is much less than those with DB2 QP priority control and No Control at the cost of the OLAP workloads. If we look at the numbers with 5 percent and 10 percent relaxation for the performance goal of the OLTP workload, Query Scheduler is much better than the other three control methods.

Table 3 Number of SLO violations for OLAP and OLTP workloads

Relaxation	QS Control			QP Priority Control			No Control		
	OLAP	OLTP	Total	OLAP	OLTP	Total	OLAP	OLTP	Total
Strict Violation	42	34	76	4	66	70	0	71	71
5% Relaxation	38	15	53	3	54	57	0	52	52
10 Relaxation	38	6	44	3	45	48	0	47	47

Table 4 Deviations from performance goals

Control Method	Class 1	Class2	Class 3
QS Control	0.178	0.169	0.031
QP Priority Control	0.315	0.231	0.034
No Control	0.343	0.186	0.035

We also calculated the deviations of the performance from their goals. The equation used is:

$$s_i = \sqrt{\frac{1}{N}\sum_{k=1}^{N}\left(g_i^k - \bar{g}_i\right)^2}$$

where s_i is the standard deviation of the service class *i*, N is the number of periods, g_i^k is the performance of the service class *i* in k^{th} period, and $\bar{g}_i$ is the performance goal of the service class *i*. As shown in Table 4, the deviations of all three service classes with Query Scheduler control are better than those with DB2 QP priority control and No Control.

In conclusion, 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 29, 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 27. Query Scheduler, however, as shown in Figure 29, 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.

5.3 Some Concerns about the Implementation

As a proof-concept implementation of the framework for workload adaptation, Query Scheduler shows our framework is effective for adapting multiple workloads to meet their SLOs. From the evaluation, however, we notice that the oscillation of performance of service classes is big, even though the workloads are stable. The oscillation is partly

due to the nature of workloads, which is not a constant, and partly for the reason that the implementation is inaccurate. Further improvement of Query Scheduler is needed to make Query Scheduler more accurate.

First of all, performance prediction largely depends on the estimation of resource demand (workload characterization). Query cost, the estimate of resource demand of a query, reflects the combined resource requirements of a query, including the resource requirements of CPU, I/O, buffer pool, sort hip, lock list, etc. Using query cost as resource demand hides the fact that same amount of query cost may have different resource usage patterns. For example, 1000 timerons may consist of 100 timeron CPU cost plus 900 timeron I/O cost for a query, and 900 timeron CPU cost plus 100 timeron I/O cost for another query. Use of detailed query costs, instead of total query cost, is preferred to improve the accuracy of performance prediction. However, DB2 QP does not acquire detailed query costs, though the DB2 Explain Facility can. The use of DB2 Explain Facility would introduce further overhead, which is much larger than that of DB2 QP. Use of detailed query costs will be left as future issue until the overhead for acquiring it is reasonable.

Performance models also play an important role in performance prediction. Because modeling a complex system like a DBMS is difficult, we use simplified queuing network models, which are inaccurate for describing the systems and introduces some errors in performance prediction. Making use of system monitoring information or measurement can contribute a lot to rectify the inaccurate prediction of a performance

model. This can be done by updating the performance model parameters or directly updating the performance prediction. We discuss this issue in the next chapter.

Admission control can introduce errors. Although class cost limits can be adjusted smoothly, the finest control one can reach is a whole query. The difference between the class cost limit and the total cost of running query of a service class changes over time. This also constitutes a part of the oscillation.

5.4 Comparison with Previous Techniques

Compared to the techniques discussed in Section 2.3, Query Scheduler has several advantages. First, compared to M & M. and PAQRS, Query Scheduler uses a performance objective encapsulation technique to manage multiple classes collectively to reflect the interdependence between classes, instead of managing SLOs individually. This allows Query Scheduler to add or remove a service class easily. Query Scheduler takes a performance model based approach to find a solution for workload control, instead of predefined heuristics, which can reflect the system dynamics.

Second, compared to ASM, Query Scheduler can directly address the resource needs for a service class by adjusting its cost limit. Since ASM allocates resources to allocation groups, resource allocation might not reflect the requirements of all the performance classes mapped into the allocation groups.

Third, compared to QoS Controller and WSWLM, Query Scheduler takes the variation of query size into consideration and uses a cost-based approach to allocate

resources, which lifts the assumption that the work requests are similar in size in order to perform admission control based on MPLs.

5.5 Summary

This chapter evaluates the effectiveness of Query Scheduler in providing differentiated service to workload classes with different SLOs. Experimental results show that Query Scheduler is able to respond to the workload changes using admission control to meet the SLOs and gives preference to important service classes, or to the service classes whose performance goals are violated. Query Scheduler can provider better quality of differentiate services than DB2 QP.

To eliminate the overhead for directly controlling OLTP workload, an approach for adapting OLTP workload through controlling OLAP workloads is proposed. Experiment results show its effectiveness.

The inaccuracy of Query Scheduler is due to inaccuracy of query cost, performance models and the nature of admission control. Chapter 6 discusses the approach to improving the accuracy of performance prediction.

Chapter 6

Improving the Accuracy of Performance Prediction

Two sources contribute to the performance prediction in the autonomic control loops of workload adaptation: performance modeling and system monitoring. Performance modeling tries to predict the performance of the target system through a model that describes the features of the target system. Modeling a complex system like a DBMS is difficult. Simplified models like queuing models are inaccurate. Making use of system monitoring information or measurements can contribute to rectifying the inaccuracy of a performance model. This can be done by updating the performance model parameters or directly updating the performance prediction.

The Kalman filter, an optimal tracking filter, estimates the state of a time-varying process by taking advantages of both the state prediction derived from the process and the state measurement [WB06]. It provides us with a framework for making use of the performance measurement to improve the prediction accuracy of a performance model.

This chapter discusses an approach to applying the Kalman filter in the framework for workload adaptation to improve the accuracy of the performance

prediction, and hence, better meet the SLOs. The rest of the chapter is organized as follows. Section 6.1 introduces the Kalman filter. Section 6.2 models the process for performance prediction as a time varying process so that a Kalman filter can be applied to it, and gives the resulting process and measurement equations. Section 6.3 discusses how to determine the parameters in order to solve the Kalman filter. An evaluation of the Kalman filter in improving the accuracy of performance prediction is presented in Section 6.4.

6.1 Kalman Filter

A Kalman filter is a tracking filter. It is a set of mathematical equations that recursively estimate the state of a process, in a way that minimizes the mean of the squared error. The Kalman filter is very powerful in that it not only supports estimations of past, present, and even future states, but also works well even when the precise nature of the modeled system is unknown.

A simple Kalman filter [WB06] tries to estimate the state, consisting of n state variables $x_1, x_2, \cdots, x_n$, of a discrete-time controlled process that is governed by the linear stochastic difference equations:

$$\begin{cases} x_1^k = a_1^{k-1} x_1^{k-1} + w_1^{k-1} \\ x_2^k = a_2^{k-1} x_2^{k-1} + w_2^{k-1} \\ \quad \cdots \\ x_n^k = a_n^{k-1} x_n^{k-1} + w_n^{k-1} \end{cases} \tag{6.1}$$

with m measurements $y_1, y_2, \cdots, y_m$, which have the relation with the state variables as follows:

$$\begin{cases} y_1^k = h_{11}^k x_1^k + h_{12}^k x_2^k + \cdots + h_{1n}^k x_n^k + v_1^k \\ y_2^k = h_{21}^k x_1^k + h_{22}^k x_2^k + \cdots + h_{2n}^k x_n^k + v_2^k \\ \cdots \\ y_m^k = h_{m1}^k x_1^k + h_{m2}^k x_2^k + \cdots + h_{mn}^k x_n^k + v_m^k \end{cases} \tag{6.2}$$

where superscripts identify the parameters in the stated steps in both sets of equations. The factors a_i^{k-1} $(i = 1, 2, \cdots, n)$ in the process equations (6.1) relate the state variable x_i $(i = 0, 1, \cdots, n)$ at the step *k-1* to the step *k*, without considering the process noise. The factors $h_{i1}^k, h_{i2}^k, \cdots, h_{in}^k$ $(i = 1, 2, \cdots, m)$ in the measurement equations (6.2) relate the state variables to the measurement y_i $(i = 0, 1, \cdots, m)$ without considering the measurement noise. The random variables w_i^{k-1} $(i = 1, 2, \cdots n)$ and v_i^k $(i = 1, 2, \cdots m)$ represent the process noise at the $(k-1)^{st}$ step and measurement noise at the k^{th} step, respectively, and are assumed to be independent, white, and with normal probability distributions with the means equal to zero.

We denote following:

State: $\mathbf{x}_k = \begin{bmatrix} x_1^k \\ x_2^k \\ \vdots \\ x_n^k \end{bmatrix}$

Measurements: $\mathbf{y}_k = \begin{bmatrix} y_1^k \\ y_1^k \\ \vdots \\ y_m^k \end{bmatrix}$

Process matrix: $\mathbf{A_k} = \begin{bmatrix} a_1^k & 0 & 0 & 0 \\ 0 & a_2^k & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & a_n^k \end{bmatrix}$

Measurement matrix: $\mathbf{H_k} = \begin{bmatrix} h_{11}^k & h_{12}^k & \cdots & h_{1n}^k \\ h_{21}^k & h_{22}^k & \cdots & h_{2n}^k \\ \vdots & \vdots & \ddots & \vdots \\ h_{m1}^k & h_{m2}^k & \cdots & h_{mn}^k \end{bmatrix}$

Process noise matrix: $\mathbf{w}_k = \begin{bmatrix} w_1^k \\ w_2^k \\ \vdots \\ w_n^k \end{bmatrix}$

Measurement noise matrix: $\mathbf{v}_k = \begin{bmatrix} v_1^k \\ v_2^k \\ \vdots \\ v_m^k \end{bmatrix}$

and $\mathbf{Q}_k$ and $\mathbf{R}_k$ the covariance matrices for $\mathbf{w}_k$ and $\mathbf{v}_k$, respectively. We have the process and measurement equations in the matrix format:

Process equation: $\mathbf{x}_k = \mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{w}_{k-1}$ (6.3)

Measurement equation $\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k$ (6.4)

The state at the step *k* can be predicted as $\tilde{\mathbf{x}}_k$ from the process equation given the state estimation $\hat{\mathbf{x}}_{k-1}$ at step *k-1*:

$$\tilde{\mathbf{x}}_k = \mathbf{A}_{k-1}\hat{\mathbf{x}}_{k-1} \tag{6.5}$$

It also can be calculated from the measurement equation by making measurement $\mathbf{y}_k$ in current step *k*. The essence of the Kalman filter is that it estimates the current state $\hat{\mathbf{x}}_k$ by striking the balance between the prediction and measurement based on both the accuracy of the process and the accuracy of the measurement. The estimation is optimal in terms of the least square error and converges quadratically. The difference between the measurement and the prediction, also known as the innovate, is

$$\mathbf{z}_k = \mathbf{y}_k - \mathbf{H}_k\tilde{\mathbf{x}}_k \tag{6.6}$$

It is used to rectify the prediction errors in order to give a good estimate:

$$\hat{\mathbf{x}}_k = \tilde{\mathbf{x}}_k + \mathbf{K}_k\mathbf{z}_k \tag{6.7}$$

where $\mathbf{K}_k$ is called the Kalman gain, which can be recursively calculated with the following equations.

$$\mathbf{K}_k = \tilde{\mathbf{P}}_k\mathbf{H}_k^T[\mathbf{H}_k\tilde{\mathbf{P}}_k\mathbf{H}_k^T + \mathbf{R}_k]^{-1} \tag{6.8}$$

$$\hat{\mathbf{P}}_k = [\mathbf{I} - \mathbf{K}_k\mathbf{H}_k]\tilde{\mathbf{P}}_k \tag{6.9}$$

$$\tilde{\mathbf{P}}_{k+1} = \mathbf{A}_k\hat{\mathbf{P}}_k\mathbf{A}_k^T + \mathbf{Q}_k \tag{6.10}$$

where the superscript *T* represents transpose of a matrix. $\hat{\mathbf{P}}_k$ and $\tilde{\mathbf{P}}_{k+1}$ are the a prior estimation error covariance and the a posteriori estimation error covariance and are

updated during the recursive calculation. The discrete algorithm for the Kalman filter is shown in Figure 30.

Initial:Given the initial estimate $\hat{\mathbf{x}}_0$, and the estimate error covariance $\hat{\mathbf{P}}_k$:

1. Predict the state

$$\tilde{\mathbf{x}}_k = \mathbf{A}_{k-1}\hat{\mathbf{x}}_{k-1}$$

2. Project the prediction error covariance

$$\tilde{\mathbf{P}}_{k-1} = \mathbf{A}_{k-1}\hat{\mathbf{P}}_{k-1}\mathbf{A}_{k-1}^T + \mathbf{Q}_{k-1}$$

3. Take measurement and calculate the innovate

$$\mathbf{z}_k = \mathbf{y}_k - \mathbf{H}_k\tilde{\mathbf{x}}_k$$

4. Calculate the Kalman gain

$$\mathbf{K}_k = \tilde{\mathbf{P}}_k\mathbf{H}_k^T[\mathbf{H}_k\tilde{\mathbf{P}}_k\mathbf{H}_k^T + \mathbf{R}_k]^{-1}$$

5. Update the estimate with measurement

$$\hat{\mathbf{x}}_k = \tilde{\mathbf{x}}_k + \mathbf{K}_k\mathbf{z}_k$$

6. Update the estimate error covariance

$$\hat{\mathbf{P}}_k = [\mathbf{I} - \mathbf{K}_k\mathbf{H}_k]\tilde{\mathbf{P}}_k$$

Figure 30 The discrete algorithm of the Kalman filter

The Kalman filter guarantees optimality when the process and measurement equations are linear. For nonlinear process and measurement equations, the extended Kalman filter [Ribe04] can be applied with sub-optimality.

6.2 Process and Measurement Equations

In the framework for workload adaptation, we use the following performance model to predict the SLOs for the next control interval and help to find an optimal resource allocation plan:

$$x_i^k = x_i^{k-1}C_i^k / C_i^{k-1}$$

where x_i^{k-1} and x_i^k are the performance (execution 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.

There are at least two sources that contribute to the inaccuracy of the model:

- Inaccuracy of the performance model:

$$x_i^k = x_i^{k-1} C_i^k / C_i^{k-1} + w_i^{k-1} \tag{6.11}$$

  where w_i^{k-1} is the noise of the performance model.

- Inaccuracy of the measurement y_i^k of x_i^k, partially due to workload fluctuation and partially due to the measurement method. This can be expressed as:

$$y_i^k = x_i^k + v_i^k \tag{6.12}$$

  where v_i^k is the measurement noise.

For the ease of presentation, we denote:

$$\mathbf{x}_k = \begin{bmatrix} x_1^k & x_2^k & \cdots & x_n^k \end{bmatrix}^T$$

$$\mathbf{y}_k = \begin{bmatrix} y_1^k & y_2^k & \cdots & y_n^k \end{bmatrix}^T$$

$$\mathbf{w}_k = \begin{bmatrix} w_1^k & w_2^k & \cdots & w_n^k \end{bmatrix}^T$$

$$\mathbf{v}_k = \begin{bmatrix} v_1^k & v_2^k & \cdots & v_n^k \end{bmatrix}^T$$

We have:

$$\mathbf{x}_k = \mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{w}_{k-1} \tag{6.13}$$

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k \tag{6.14}$$

where

$$\mathbf{A}_k = \begin{vmatrix} C_1^k / C_1^{k-1} & 0 & \cdots & 0 \\ 0 & C_2^k / C_2^{k-1} & 0 & \vdots \\ \vdots & 0 & \ddots & 0 \\ 0 & \cdots & 0 & C_n^k / C_n^{k-1} \end{vmatrix}, \ \mathbf{H}_k = \mathbf{I}.$$

Taking equations (6.13) and (6.14) as the process and measurement equations, respectively, we can apply the Kalman filter to our framework for workload adaptation. That is to say, we can make a better estimate of the performance at the current control interval by considering both the predicted and measured performance for the current control interval, so that the predicted performance for the next control interval is more accurate. In the next section, we discuss the roles that the parameters $\mathbf{Q}_k$ and $\mathbf{R}_k$ play in the Kalman filter and how to determine them.

6.3 Parameter Determination

To use the Kalman filter, we need to determine the initial estimation error covariance $\tilde{\mathbf{P}}_0$, the process error covariance $\mathbf{Q}_k$ and the measurement error covariance $\mathbf{R}_k$. Before we discuss how to determine them, let us first find out how these parameters affect the Kalman filter. Intuitively, the larger the elements in $\mathbf{Q}_k$, the more inaccurate the process

and the smaller the contribution to the estimate by the prediction. The larger the elements in $\mathbf{R}_k$, the more inaccurate the measurement and the smaller the contribution to the estimate by the measurement. The same observation can be derived from the equations (6.10) and (6.8). A large $\mathbf{Q}_k$ results in large estimation error covariance and suggests that the measurement is more accurate. A large $\mathbf{R}_k$ produces a small Kalman gain which puts less weight on the innovate and suggests that prediction is more accurate. A good Kalman filter can be obtained by carefully tuning $\mathbf{Q}_k$ and $\mathbf{R}_k$.

$\tilde{\mathbf{P}}_0$ is the initial estimation error covariance. Because the Kalman filter converges quickly [WB06], it is does not matter how the elements in $\tilde{\mathbf{P}}_0$ are chosen as long as the elements at the main diagonal are not zero. $\mathbf{Q}_k$ and $\mathbf{R}_k$ vary from interval to interval and should be determined dynamically, which is difficult. In practice, both $\mathbf{Q}_k$ and $\mathbf{R}_k$ are supposed to be diagonal and determined prior to operation [Ribe04]. If a process can be observed directly, which is true for our case, we can calculate the process error covariance $\mathbf{Q}_k$ beforehand by taking some off-line sample observations. In most cases, it is not possible to directly observe the process, and a simple process noise model with enough uncertainty can be used as the process error covariance $\mathbf{Q}_k$. The measurement error covariance $\mathbf{R}_k$, on the other hand, generally can be determined beforehand by taking some off-line sample measurements.

Even we can directly observe the process, it is difficult to accurately determine the process error covariance $\mathbf{Q}_k$. This is because the measurement matrix $\mathbf{A}_{k-1}$ varies

from interval to interval, which makes it impossible to observe the process with every single $\mathbf{A}_{k-1}$. However, we can approximately determine the process error covariance $\mathbf{Q}_k$ regardless the variation of the measurement matrix $\mathbf{A}_{k-1}$ with some off-line samples. Suppose we have L predicates ($x_i^k, k = 1, 2, \cdots, L$) and measures ($y_i^k, k = 1, 2, \cdots, L$). The difference between the predicted value and the measured value ($x_i^k - y_i^k, k = 1, 2, \cdots, L$) roughly reflects the errors resulting from the process, so we calculate the diagonal elements q_{ii} of process noise covariance $\mathbf{Q}_k$ as the variance of the difference, which can be formulated as:

$$q_{ii} = \frac{1}{L}\sum_{k=1}^{L}\left((x_i^k - y_i^k) - \frac{1}{L}\sum_{k=1}^{l}(x_i^k - y_i^k)\right)^2 . \tag{6.15}$$

Now we have the fixed process noise covariance $\mathbf{Q}_k$ calculated by off-line samples. We can also dynamically calculate the process noise covariance $\mathbf{Q}_k$ by applying the equation with the nearest L historical predicates and measures. To reduce the computation complexity, we can simply use the predication and the measure in current control interval to calculate the diagonal elements q_{ii} :

$$q_{ii} = (x_i^k - y_i^k)^2 . \tag{6.16}$$

When it comes to calculate the measurement noise covariance $\mathbf{R}_k$, it is important to know that the accuracy of a measure depends on the number of queries measured. The more queries the accurate the measure. As per the central limit theorem, the diagonal

element r_{ii} of measurement noise covariance $\mathbf{R}_k$ is inversely proportional to the number of queries measured (N):

$$r_{ii} = Constant / N. \tag{6.17}$$

If we know N^*, the number of queries needed to reach the predefined measurement accuracy, for example 95% confidence interval of ±5% measurement y_i^k, then we have:

$$r_{ii}^* = \left(0.5 y_i^k / 1.96 \right)^2 = Constant / N^*.$$

$$Constant = \left(0.5 y_i^k / 1.96 \right)^2 N^*. \tag{6.18}$$

Through experiments we determined that $N^* = 1402$. Combining the result with equations (6.17) and (6.18), we have:

$$r_{ii} = 0.912 \left(y_i^k \right)^2 / N. \tag{6.19}$$

If workload is relatively stable, $\mathbf{R}_k$ can be determined through experiments as a fixed parameter. Otherwise, it can be calculated dynamically using the equation (6.19).

6.4 Evaluation

In order to examine the ability of the Kalman filter to improve the performance prediction accuracy, we designed a set of experiments with both deterministic and random workloads. Section 6.4.1 presents the metrics for measuring the improvement of the

performance prediction accuracy. Section 6.4.2 gives the experimental settings, including the system environments, workloads and the choice of performance metrics for the workloads. Section 6.4.3 describes the set of experiments. The results of the experiments and the analysis of the results are given in Section 6.4.4.

6.4.1 Improvement Metrics

The improvement is measured by the performance prediction error and the percentage unpredicted SLO violation. The performance prediction errors are calculated as the root-mean-square (RMS) of *n* prediction errors using the following equation:

$$RMS = \sqrt{1/n \sum_{k=1}^{n} (x_i^k - y_i^k)^2}$$

There are two types of SLO violations: predicted SLO violations and unpredicted SLO violations. A predicted SLO violation is a violation that has been predicted by the system due to heavy workload. A predicted SLO violation is unavoidable and inherent, and hence, should not be included when measuring the improvement. An unpredicted SLO violation is a SLO violation that the system did not predict due to the inaccuracy of the performance model, and should be accounted for when measuring the improvement. The percentage of unpredicted SLO violations is calculated as the number of control intervals with unpredicted SLO violations divided by the total number of control intervals.

6.4.2 Experimental Settings

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 Version 8.2 and Query Patroller as supporting components.

We use the TPC-H standard DSS benchmark as our test database and workload. The database consists of 300MB of data. The workload consists of two classes of TPC-H queries submitted by interactive clients or batch jobs one after another with zero think time. Each class has a SLO consisting of a performance goal and an importance level. Class 0 is deemed more important than Class 1 by assigning a higher importance level. The performance goals are determined experimentally with the total cost limit 300,000 timerons to ensure that they are achievable.

Workload intensity is controlled by the number of clients or batch jobs for each class. We divide the tests into 6 periods with different workload intensity. Each period consists of sixteen 5-minute intervals. For the deterministic workload, the number of clients in each period is fixed (Figure 31). Period 1 is the lightest period with 15 clients from both classes submitting queries simultaneously. Period 3 is the heaviest period with 25 clients from the high importance class 1 and 15 clients from the low importance class 2 submitting queries simultaneously. The order from light to heavy for the rest of the periods is: 4, 6, 2, and 5. For the random workload, we allow a maximum workload intensity change of 10% based on the deterministic workload. This setting simulates both small workload fluctuations within each period and major workload changes between

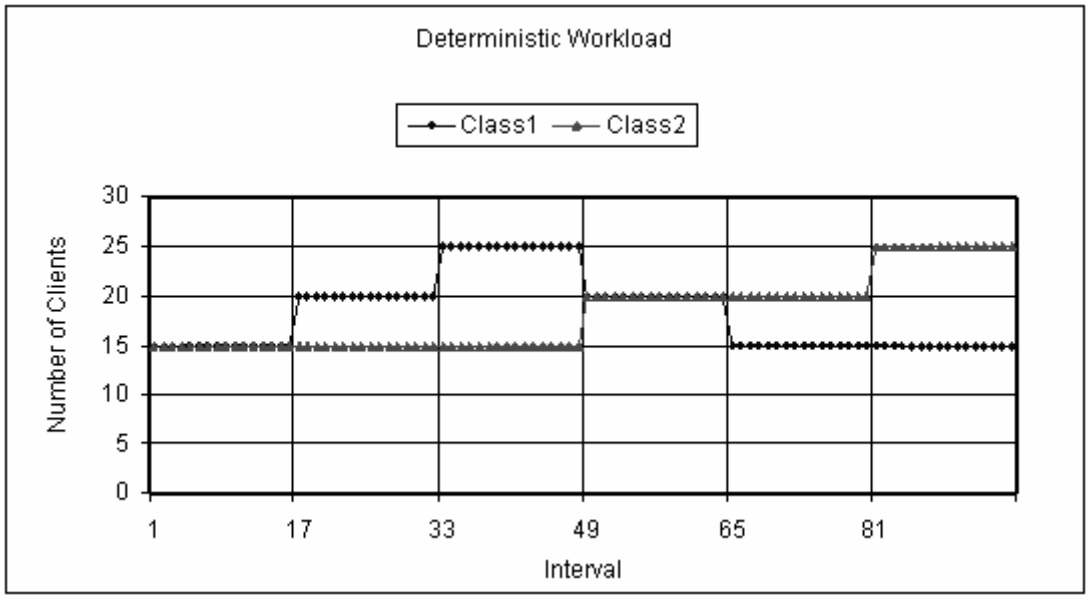

Figure 31 Deterministic workload

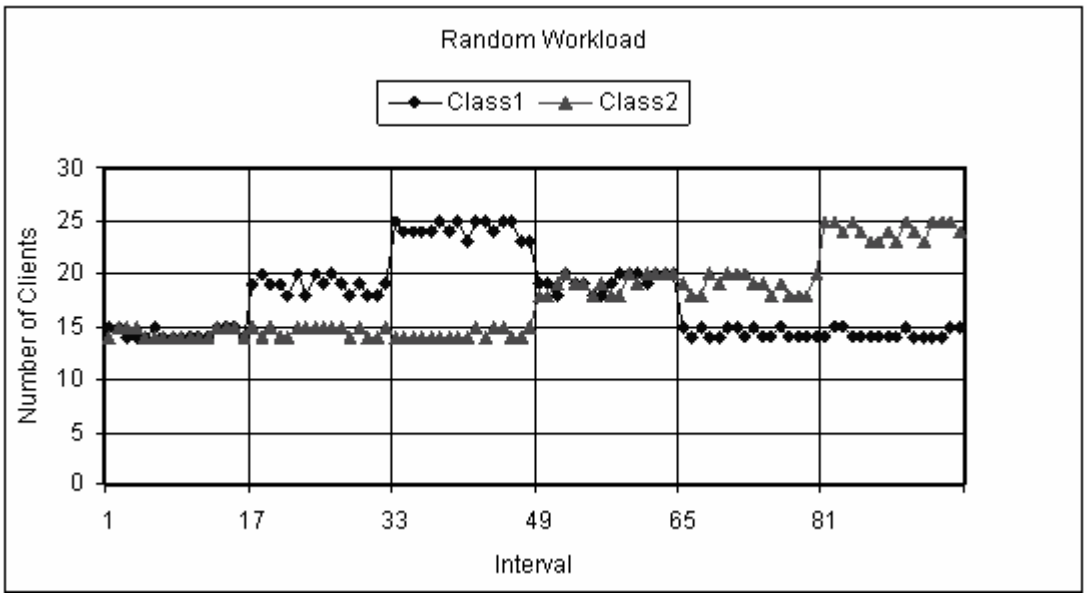

Figure 32 Random workload

periods. Figure 31 shows the deterministic workload and Figure 32 is a sample random workload.

We use query velocity as the performance metric. The performance goals for class 1 and class 2 are set as 0.55 and 0.45 respectively. They are carefully set such that the total system cost limit is in effect all the time to ensure the system is in a consistent state.

6.4.3 Experiments

The following set of experiments evaluates the improvement made by applying the Kalman filter. We compare three scenarios in the experiments.

No Kalman Filter

This scenario does not use the Kalman filter and serves as our baseline measure to observe the improvement of accuracy of performance prediction. We collect the prediction and measurement data to calculate the improvement metrics described in Section 6.4.1, and the prediction error covariance and measurement error covariance for the next experiment.

Kalman Filter with Fixed Parameters

This scenario incorporates the Kalman filter with the pre-calculated prediction error covariance and the measurement error covariance from the previous scenario, and is designed as the scenario with a priori knowledge of the workloads. The prediction error covariance is calculated by applying Equation (6.15) for each period and the

measurement error covariance is calculated by using Equation (6.19) with y_i^k replaced by the average of the measures in each period. The improvement metrics are calculated.

Kalman Filter with Dynamic Parameters

This scenario incorporates the Kalman filter with the prediction error covariance and the measurement error covariance calculated on the fly using Equations (6.16) and (6.19), respectively, in each interval and is served as the scenario without a priori knowledge of the workloads. The improvement metrics are calculated.

All three scenarios are run with both the deterministic workloads and the random workloads described in the Section 6.4.2. For the deterministic workloads, Table 5 shows the diagonal elements of prediction error covariance (q_{00} and q_{11}) and the measurement error covariance (r_{00} and r_{11}) calculated from the case with no Kalman filter. Figures 33 and 34 show the prediction errors (RMS) and the percentage unpredicted SLO violation, respectively. 90% Confidence interval values are given on the tops of each bar. For the random workloads, Table 6 shows the diagonal elements of prediction error covariance (q_{00} and q_{11}) and the measurement error covariance (r_{00} and r_{11}) calculated from the base scenario. Figures 35 and 36 show the prediction errors (RMS) and the percentage unpredicted SLO violation, respectively. 90% Confidence interval values are given on the tops of each bar.

Table 5 Prediction error covariance and measurement error covariance for deterministic workload

	q_{00}	q_{11}	r_{00}	r_{11}	q_{00}/r_{00}	q_{11}/r_{11}
Period1	0.0051	0.0036	0.0025	0.0022	2.0610	1.6359
Period2	0.0048	0.0012	0.0017	0.0016	2.9018	0.7641
Period3	0.0022	0.0016	0.0012	0.0013	1.7653	1.2041
Period4	0.0039	0.0044	0.0016	0.0010	2.4458	4.5655
Period5	0.0046	0.0019	0.0024	0.0014	1.9262	1.4249
Period6	0.0023	0.0030	0.0022	0.0010	1.0509	3.1200
Average	0.0037	0.0025	0.0019	0.0013	2.0252	2.1191

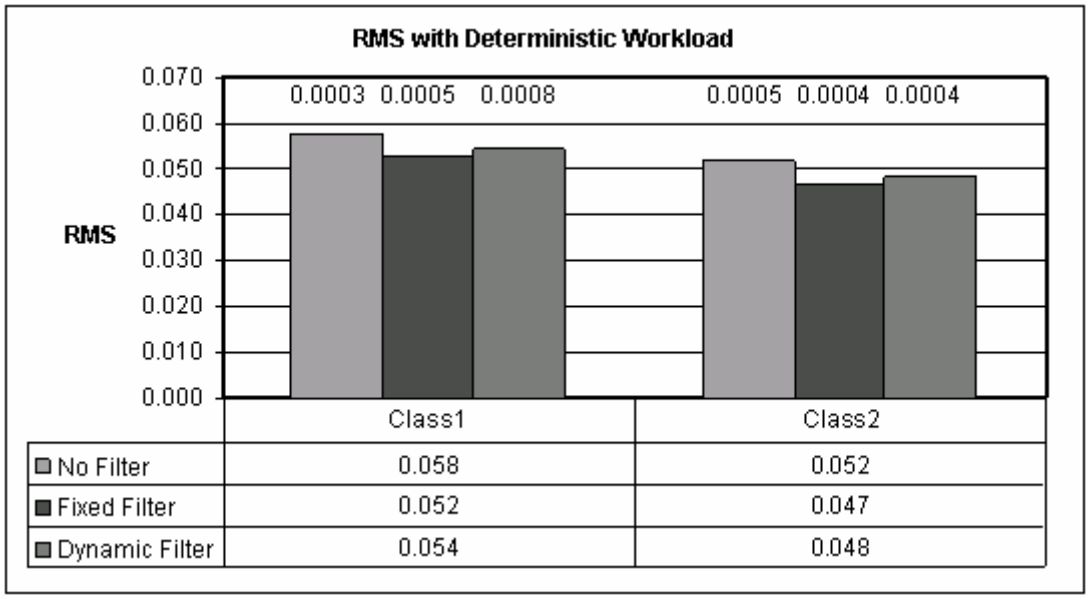

Figure 33 Prediction errors (RMS) with deterministic workload

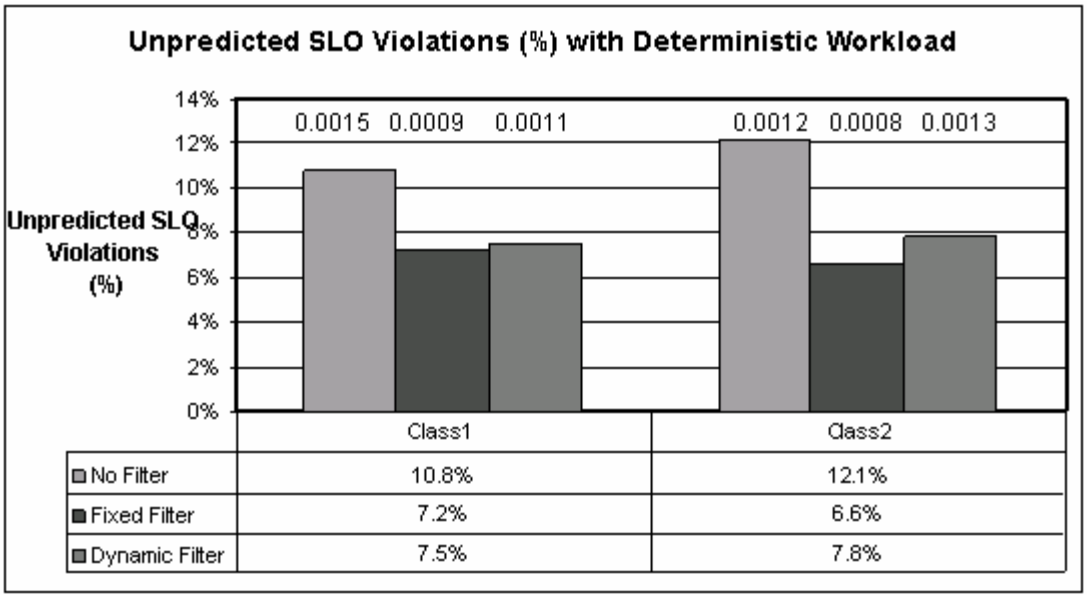

Figure 34 Percentage unpredicted SLO violation with deterministic workload

Table 6 Prediction error covariance and measurement error covariance for random workload

	q_{00}	q_{11}	r_{00}	r_{11}	q_{00}/r_{00}	q_{11}/r_{11}
Period1	0.0027	0.0051	0.0026	0.0023	1.0385	2.2174
Period2	0.0024	0.0030	0.0017	0.0019	1.4118	1.5789
Period3	0.0029	0.0039	0.0014	0.0014	2.0714	2.7857
Period4	0.0023	0.0032	0.0018	0.0011	1.2778	2.9091
Period5	0.0045	0.0035	0.0024	0.0015	1.8750	2.3333
Period6	0.0037	0.0017	0.0024	0.0010	1.5417	1.7000
Average	0.0030	0.0032	0.0020	0.0015	1.5360	2.2541

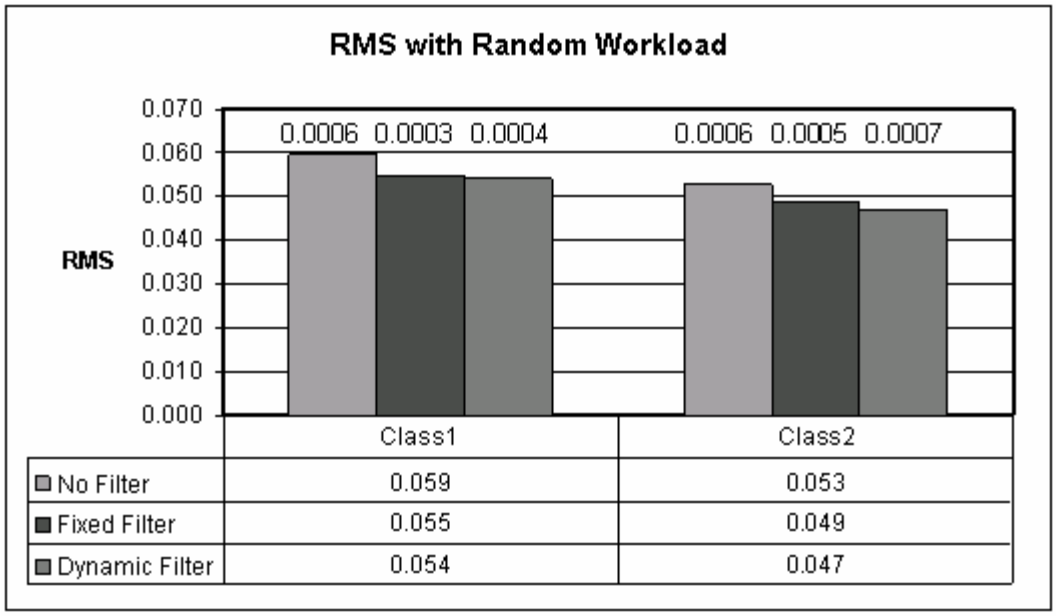

Figure 35 Prediction errors (RMS) with random workload

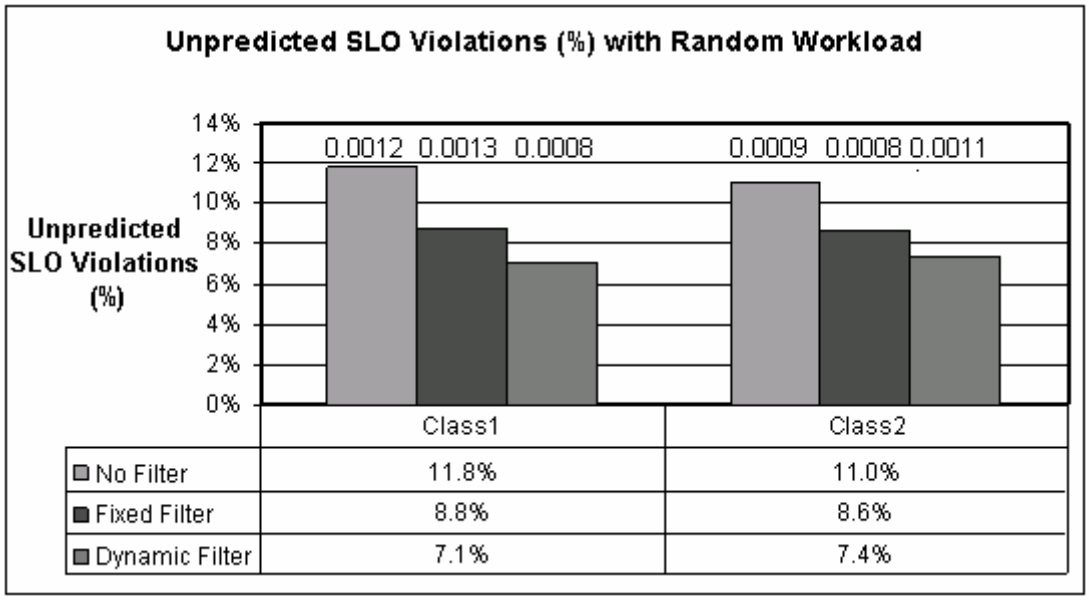

Figure 36 Percentage unpredicted SLO violation with random workload

6.4.4 Analysis of the Experiment Results

Measurement vs. Prediction

From Tables 5 and 6 we can see that except in the period 1 for Class 2, the measurement error covariance is consistently smaller than the prediction error covariance in every other period ($q_{00} > r_{00}, q_{11} > r_{11}$). This suggests that the measurement is more accurate than prediction. We also notice that the prediction error covariance changes randomly. For Class 1 the average prediction error covariance for deterministic workload is larger than that for random workload. For class 2, however, average prediction error covariance for deterministic workload is smaller than that for random workload. This justifies our choice to calculate the prediction error covariance using Equation (6.16) for the dynamic Kalman filter.

Prediction Error

The RMS of prediction errors is decreased after applying the Kalman filter. As shown in Figures 33 and 35, the RMS of prediction errors without the Kalman filter is greater than that with the Kalman filter. For the deterministic workloads, the fixed Kalman filter produces a smaller RMS of prediction errors than the dynamic Kalaman filter. There are two reasons for this result. First, the fixed Kalman filter possesses a priori knowledge of the workloads and the dynamic Kalman filter does not. Second, the workloads are stable in each period, which allows the fixed Kalman filter to have better parameter estimates than the dynamic Kalman filter by averaging all the predictions and measures in each period. The percentage reduction for the dynamic Kalman filter is

$(0.058-0.054)/0.058=6.9\%$ for Class 1 and $(0.052-0.048)/0.052=7.6\%$ for Class 2. This means there is at least 6.9% improvement of prediction accuracy for the deterministic workload after applying the Kalman filter.

For the random workloads, the dynamic Kalman filter produces a smaller RMS of prediction errors than the fixed Kalaman filter. This is because the fixed Kalman filter can not catch the dynamic characteristics of workloads. The parameter average of the fixed Kalman filter hides randomly changing property of the parameters. The percentage reduction for the fixed Kalman filter is $(0.059-0.055)/0.059=6.8\%$ for Class 1, and $(0.053-0.049)/0.053=7.5\%$ for Class 2. This means there is at least a 6.8% improvement of prediction accuracy for the random workload after applying the Kalman filter.

Number of Unpredicted SLO Violations

The percentage unpredicted SLO violations is also decreased after applying the Kalman filter as shown in Figures 34 and 36. For the deterministic workloads the fixed Kalman filter outperforms the dynamic Kalman filter. For the random workloads the dynamic Kalman filter is more effective than the fixed Kalman filter. This also supports the explanations provided when we discussed the RMS of the prediction errors.

Fixed Kalman Filter vs. Dynamic Kalman Filter

From the previous analysis we can see that the dynamic Kalman filter is better than the fixed Kalman filter when workloads change randomly, and vice versa when workloads are relatively stable. Both Kalman filters can improve the accuracy of performance

prediction. They decrease the RMS of prediction errors and the percentage unpredicted SLO violation.

6.5 Summary

This chapter studies an approach to improving the accuracy of performance prediction using a Kalman filter. A Kalman filter is a tracking filter, which tracks the state changes of a time varying process. The discrete algorithm for the Kalman filter shown in Figure 30 is the basis for applying it into real problem. We formulated the process and measurement equations for improving the accuracy of performance prediction by using the Kalman filter. The parameters, including the initial estimation error covariance, the process error covariance and the measurement error covariance, have a significant influence on the performance of the Kalman filter. We discussed the approaches to determining the parameters for the Kalman filter. The process error covariance and the measurement error covariance can be calculated offline as fixed parameters or online dynamically. The experiments with both deterministic and random workloads show that the Kalman filter can improve the accuracy of performance prediction. The Kalman filter with dynamically calculated parameters outperforms the Kalman filter with pre-calculated fixed parameters when workloads change randomly, and vice versa when workloads are stable.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis we study workload adaptation in autonomic DBMSs. When we review workload management techniques, we see a clear trend that workload management is evolving from resource-oriented workload management into performance-oriented workload management. The style of workload management has also changed from offline analysis to online adaptation.

As an autonomic technique for workload management, workload adaptation must be able to detect workload changes and make workload control plans. Workload changes can be detected through workload characterization and/or performance monitoring. Workload control plans can be derived using four approaches, namely, a performance model approach, a statistical approach, a heuristic approach, and a threshold approach.

We proposed a general framework for workload adaptation. It consists of two processes, namely a workload detection process and a workload control process. The workload detection process detects workload changes and provides knowledge about the workload. The workload control process reacts to the workload changes by performing

resource allocation, parameter tuning and admission control to maintain system service levels. The framework also involves four functional components, namely workload characterization, performance modeling, system monitoring, and workload control. Workload characterization is concerned with measuring and modeling production workloads. Performance modeling tries to predict the performance of the target system through a model that describes the features of the target system. System monitoring, or feedback, tells how well the system is performing by continuously acquiring the execution information of the workload and the resource usage of the system. Workload control components try to find and enforce an optimal resource allocation plan to meet the management goals.

In order to prove the effectiveness of the framework, we implemented Query Scheduler, which is a prototype of the framework 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 an objective function that encapsulates the SLOs with utility functions. Through experiments we have shown the effectiveness of the framework.

We proposed a general form of utility functions, which encapsulates the workload importance and the performance goal of a workload, and describes how well a system meet its SLO. Both response time goals and velocity goals can be applied consistently.

We showed that Query Scheduler can directly control OLAP workloads towards their performance goals. However, the overhead to directly control OLTP workloads makes Query Scheduler impractical to directly manipulate OLTP queries. We therefore

developed a technique for enforcing indirect control over OLTP workload by directly controlling the OLAP workloads. This technique enables Query Scheduler to control the mixed workloads consist of both OLAP and OLTP queries to meet their performance goals.

The effectiveness of workload adaptation depends on the accuracy of performance prediction. We present an approach for improving the accuracy of performance prediction to better meet the SLOs. In this approach, we use a tracking filter, Kalman filter, to track performance changes, so that the performance prediction is always based on a more accurate estimate of the current state. Experiments show that this approach is effective. The accuracy of performance prediction is improved and the percentage unpredicted SLO violation drops.

In conclusion, this study makes four contributions to workload management. The first contribution is a general framework for performance-oriented workload adaptation in autonomic DBMSs. The second contribution is a prototype implementation of the framework, called Query Scheduler, which adapts the workload for an instance of DB2. The third contribution is a cost-based performance model, which guides the making of workload control plans. The fourth contribution is the introduction of tracking techniques to improve the accuracy of performance prediction. These contributions together constitute an effective framework for workload adaptation in autonomic DBMSs.

7.2 Future Work

While effective, there are still many issues that need to be addressed. One of them is quantifying workload importance. When users define a SLO, the performance goal can be objectively determined by experiments. Workload importance, however, is a subjective performance metric. It is necessary to develop a well defined process to quantify the workload importance.

The overhead of controlling OLTP workloads is another issue to be addressed. The most effective way to manage performance of OLTP workload is to directly control it. A possible approach is to implement the control mechanism inside the DBMS itself. DB2 Viper 2, a recent new version, has implemented the service class concept into its engine, which is originally implemented outside DB2 engine in DB2 QP. A recent declassified document [Bird07] shows that "In order to handle the higher volumes with minimal overhead and to provide the degree of control and monitoring desired, we needed to incorporate any new WLM technology directly into the DB2 engine infrastructure. The approach of using an auxiliary application to provide workload management for DB2, such as used by Query Patroller, simply would not be able to provide what was needed." This is a natural path for incorporating workload adaptation into database engines. When direct control over OLTP workload is feasible, performance modeling for OLTP workload will be an issue that needs to be addressed.

Cost-based resource allocation is not accurate. On the one hand, a class cost limit sets an upper-bound for a service class and is seldom reached. The resources allocated to

a service class are usually underutilized, causing performance fluctuation when query size varies widely. On the other hand, query cost is a general estimate of the usage of various resources, at least including CPU, I/O, buffer pool, sort heap, and lock list. Different types of queries have different resource usage patterns. Query cost hides this fact, which may cause inaccuracy. The possible approaches to address this issue can be studying the approaches to reducing the difference between the class cost limit and the water marks of the total cost of a service class, and using detailed query costs instead of total query cost. Finally, estimating the detailed query costs will be a challenge in the foreseeable future.

Bibliography

[ABHG07] M. Amirijoo, P. Brännström, J. Hansson, S. Gunnarsson, and S. H. Son. "Toward Adaptive Control of QoS-Importance Decoupled Real-Time Systems", *IEEE International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, Germany, 2007.

[Adam01] G. Adam. "Understanding Workload Manager: Basic Metrics", 2001, http://www.naspa.com.

[Ande73] A. Anderberg. *Cluster Analysis for Applications*, New York: Academic, 1973.

[Ange02] B. Angell. "Autonomic Computing: The Evolution Continues",2002, http://www.itworld.com/nl/db_mgr/07012002/

[Avri03] M. Avriel. *Nonlinear Programming: Analysis and Methods*, Dover Publishing, 2003.

[Ball02] C. Ballinger. "Introduction to Teradata's Priority Scheduler", 2002, http://www.teradatalibrary.com/pdf/eb3092.pdf.

[Berk02] P. Berkhin. "Survey of Clustering Data Mining Techniques", *Technical Report of Accrue Software*, http://www.ee.ucr.edu/~barth/EE242/clustering_survey.pdf.

[Binm83] K. G. Binmore. *Calculus*, Cambridge University Press, 1983, pp. 156.

[Bird07] Paul Bird. *DB2 Viper 2: What's New in Workload Management*, March, 2007.

[BMCL94] 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.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*, Cambridge University Press, 2004.

[CA02] C. H. Crawford, and A. Dan. “eModel: Addressing the Need for a Flexible Modeling Framework in Autonomic Computing”, *10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'02)*, Oct. 11 - 16, 2002, Fort Worth, TX, USA, pp. 203 - 208.

[Can93] F. Can. “Incremental Clustering for Dynamic Information Processing”, *ACM Transaction on Information Systems*, Vol. 11, No. 2, Apr. 1993, pp. 143–164.

[CCFM04] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. “Incremental Clustering and Dynamic Information Retrieval”, *SIAM Journal on Computing*, Vol. 33, No. 6, 2004, pp. 1417 - 1440.

[CG90] G. Carpenter, and S. Grossberg. “ART3: Hierarchical Search Using Chemical Transmitters in Self-organizing Pattern Recognition Architectures”, *Neural Networks*, Vol. 3, 1990, pp. 129–152.

[CTYB01] S. Castro, N. Tezulas, B. Yu, J. Berg, H. Kim, and D. Gfroerer. “AIX 5L Workload Manager”, *IBM Redbooks*, June 2001.

[DEFH03] 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.

[DHBA02] D. H. Brown Associate, Inc. “HP Raises the Bar for UNIX Workload Management”, 2004, http://whitepapers.silicon.com/0,39024759,60104905p-39000654q,00.htm.

[DHS01] R. Duda, P. Hart, and D. Stork. *Pattern Classification*, 2nd edition, New York: Wiley, 2001.

[DO74] B. Duran, and P. Odell. *Cluster Analysis: A Survey*, New York: Springer-Verlag, 1974.

[ECW01] H.M. El-Sayed, D. Cameron, and C. M. Woodside. “Automation Support for Software Performance Engineering”, *Proceedings of Joint International Conference on Measurement and Modeling of Computer Systems (SIGMetrics 2001/Performance 2001)*, June 16 - 20, 2001, Cambridge, MA, USA, pp. 301-311.

[ELL01] B. Everitt, S. Landau, and M. Leese. *Cluster Analysis*, London: Arnold, 2001.

[FB93] J. Douglas Faires, and Richard L. Burden. “Numerical Methods”, PWS-Kent Pub. Co., Boston, 1993. [GC03] A. G. Ganek and T. A. Corbi. “The Dawning of the Autonomic Computing Era”, *IBM SYSTEMS JOURNAL*, Vol. 42, No. 1, 2003.

[Fish87] D. Fisher. “Knowledge Acquisition Via Incremental Conceptual Clustering”, *Machine Learning*, Vol. 2, Issue 2, Sept. 1987, pp. 139–172.

[GC03] A. G. Ganek and T. A. Corbi. “The Dawning of the Autonomic Computing Era”, *IBM Systems Journal*, Vol. 42, No. 1, 2003.

[Hart75] J. Hartigan, *Clustering Algorithms*, New York: Wiley, 1975.

[He99] Q. He. “A Review of Clustering Algorithms as Applied to IR,” University of Illinois at Urbana-Champaign, *Technical Report*, UIUCLIS-1999/6+IRG, 1999.

[HP03] D. Hornby, and K. Pepple. *Consolidation in the Data Center: Simplifying IT Environments to Reduce Total Cost of Ownership*, Prentice Hall PTR. Jan. 24, 2003.

[HP04] HP. “HP-UX Workload Manager Overview”, 2004, http://h30046.www3.hp.com/uploads/infoworks/5982-4354EN.pdf.

[IBM03A] IBM Corporation. *MVS Planning: Workload Management*, 7th edition, Oct. 2003.

[IBM03B] IBM Corporation. *DB2 Query Patroller Guide: Installation, Administration, and Usage*, 2003.

[IBM04] IBM Corporation. *z/OS V1R6.0 RMF Performance Management Guide*, 4th edition, Sept. 2004.

[Jain91] R. Jain. “The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling”, Wiley-Interscience, New York, NY, Apr. 1991.

[JD88] A. Jain, and R. Dubes. *Algorithm for Clustering Data,* Prentice Hall, 1988.

[JMF99] A. Jain, M. Murty, and P. Flynn. “Data Clustering: A Review”, *ACM Computing Surveys*, Vol. 31, Issue 3, Sept. 1999, pp. 264 - 323.

[JTZ04] D. Jiang, C. Tang, and A. Zhang. “Cluster Analysis for Gene Expression Data: A Survey”, *IEEE Transaction on Knowledge Data Engineering*, Vol. 16, Issue 11, Nov. 2004, pp. 1370 - 1386.

[Kalm60] R. E. Kalman. “A New Approach to Linear Filtering and Prediction Problems”, *Transactions of ASME, Journal of Basic Engineering*, Vol. 82, Mar. 1960, pp. 34 - 45.

[LD86] 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.

[Ligh04] S. Lightstone. “Autonomic DB2: simplicity, performance, availability”, *IDUG 2004 - North America*, May 9 - 13, 2004, Marriott World Center, Orlando, Florida.

[LZGS84] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. “*Quantitative System Performance: Computer System Analysis Using Queuing Network Models*”, Prentice-Hall, 1984.

[MA98] D. A. Menasce, and V. A. F. Almeida. "*Capacity Planning for Web Performance: Metrics, Models, and Methods*", Prentice Hall, Upper Saddle River, NJ, 1998.

[MAFM99] D. A. Menasce, 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.

[MB03] D. A. Menasce, 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.

[MBD01] 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.

[MC79] J. J. More, and M. Y. Cosnard. "Numerical Solution of Nonlinear Equations", *ACM Transactions on Mathematical Software*, Volume 5, Number 1, Mar. 1979, pp 64 - 85.

[MCWV99] R. McDougall, A. Cockcroft, B. Wong, E. Vargas, E. Hoogendoon, T. Bialaski, and J. Johnstone. "Solaris Resource Management", 1999, http://www.sun.com/blueprints.

[Moor89] B. Moore. “ART1 and Pattern Clustering”, *Proceedings of 1988 Connectionist Models Summer School,* 1989, Morgan Kaufmann, San Mateo, CA, USA, pp. 174-185.

[NH94] R. T. Ng and J. Han. “Efficient and Effective Clustering Methods for Spatial Data Mining”, *Proceedings of the 20th International Conference on Very Large Data Bases,* Santiago, Chile, 1994, pp. 144 – 155.

[NMPH06] 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.

[PCL95] 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.

[PSTY05] 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.

[Ribe04] M. I. Ribeiro. “Kalman and Extended Kalman Filters: Concept, Derivation and Properties”, http://omni.isr.ist.utl.pt/~mir/pub/kalman.pdf.

[SCH75] J. R. Slagle, C. L. Chang, and S. R. Heller. “A clustering and Data-Reorganizing Algorithm”, *IEEE Transaction on System, Man, and Cybernetics*, Vol. 5, 1975, pp. 125–128.

[SHIN06] 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.

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

[WB06] G. Welch, and G. Bishop. “An Introduction to the Kalman Filter”, Apr. 2006, http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf

[Wood89] M. Woodside, “Throughput Calculation for Basic Stochastic Rendezvous Networks”, *Performance Evaluation*, Vol. 9, Issue 2, Apr. 1989, pp. 143-160.

[Wood02] M. Woodside. “Techniques for Deriving Performance Models from Software Designs”, Aug., 2002. http://www.sce.carleton.ca/faculty/woodside/woodside_pub.shtml.

[XW05] R. Xu, and D. Wunsch II. “Survey of Clustering Algorithms”. *IEEE Transactions on Neural Networks*, Vol. 16, No. 3, May 2005, pp. 645 – 678.

[XWP03] J. Xu, M. Woodside, and D. Petriu. “Performance Analysis of a Software Design Using the UML Profile for Schedulability, Performance, and Time”, *Proceedings of 13th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS 2003)*, Urbana, Illinois, USA, Sept 2003, pp 291 - 310.

[ZW03] T, Zheng, and M. Woodside, “Heuristic Optimization of Scheduling and Allocation for Distributed Systems with Soft Deadlines”, *Proceedings of 13th International Conference on Computer Performance Evaluation, Modelling Techniques, and Tools (TOOLS 2003)*, Urbana, Illinois, USA, Sept. 2003, pp 169-181.