Quality of Service-Enabled Management of Database Workloads

Stefan Krompass[‡] Andreas Scholz[‡] Martina-Cezara Albutiu[‡]
Harumi Kuno[§] Janet Wiener[§] Umeshwar Dayal[§] Alfons Kemper[‡]

[‡]Technische Universität München, Munich, Germany
{albutiu,kemper,krompass,scholza}@in.tum.de

[§]Hewlett-Packard Laboratories, Palo Alto, CA, USA
{firstname.lastname}@hp.com

Abstract

Database administrators struggle when managing workloads that have widely different performance requirements. For example, the same database may support short-running OLTP queries and batch jobs containing multitudes of queries with varying complexity. Different workloads may have different performance requirements, expressed in terms of service level objectives (SLOs) that must be fulfilled in order to keep the issuing database users satisfied. In this paper, we identify basic query classes and describe the challenges they pose for SLO-aware workload management. Additionally, we propose a generic architecture for an SLO-aware DBMS. We give an overview of workload management techniques already implemented in today's DBMS and outline future research directions for as yet unsupported concepts.

1 Introduction

Imagine you are a database system administrator for a large company. Your job is to administer a variety of workloads running on the database. These workloads are submitted by different customers who have unique requirements. The company's Web front end produces an OLTP-style workload with short-running parameterized queries that must be processed quickly in order to provide immediate feedback to the customers. Depending on the customer, the queries have varying importance and performance requirements. While processing the OLTP queries, your database must also handle business intelligence (BI) workloads. For example, sales managers submit analytic batch workloads to prepare financial reports for a meeting with your company's CEO. These workloads have a hard deadline and partial results are worthless. To complicate matters, members of the marketing team simultaneously need to execute complex custom queries in order to craft their new campaign.

In today's databases, OLTP and BI workloads are usually kept separate: OLTP workloads are submitted to and processed by operational databases, and BI workloads by data warehouses. For the management of each individual workload, you as the database administrator must address a variety of problems: First, you need concrete metrics that describe the customers' expectations in a way you can measure. For example, you cannot measure whether or not you meet the customer's vague expectation of a "short response time", but you can measure the elapsed time needed to respond to a query. Second, you need policies to manage incoming queries. For example, you must decide whether or not to admit a new query when you expect the query to have a negative impact on concurrent queries. Third, you need workload management policies that consider the characteristics of the workloads as a whole. In particular, you need workload management techniques to address questions like:

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

- What is the number of concurrent queries in the database system that optimizes throughput for a particular workload? What if multiple workloads are running simultaneously?
- Can the queries be scheduled according to their priorities? How should the priorities be determined?
- How long should you wait before killing an unexpectedly long-running query that hogs resources? How can you tell that the query hogs resources? What if this query is business-critical and must be completed?

Increasingly, we are seeing trends towards "operational data stores" or "operational BI", where mixed workloads run on the same database. The parallel execution of workloads with different requirements on the same database poses new challenges and requires an integrated approach for workload management. As a preparatory work, this paper focuses on workload management techniques for separate execution of the different workload types. Workload management for operational data stores is ongoing work in our research collaboration.

The rest of this paper is organized as follows: Section 2 characterizes the workloads we focus on in this paper. Section 3 defines service level objectives for different workload classes. Section 4 describes how workload management is used to meet service level objectives. Section 5 summarizes prior art in industry and academia. Our proposed solution for managing OLTP workloads is presented in Section 6. Section 7 describes challenges in workload management for BI workloads. Finally, Section 8 summarizes the contents of the paper.

2 Characterization of Workloads

Table 1 characterizes business intelligence (BI) and OLTP workloads. The vertical axis describes how queries are generated. The structure of *canned* queries is fixed; the only variety stems from the parameterization of constants. This kind of query is typically issued by software clients, often by using prepared statements. In contrast, the structure of *ad hoc* queries is not known in advance, and may vary widely. Ad hoc queries may cause performance issues because they are often only executed once and thus may not be as thoroughly optimized as canned queries.

	interactive	**batch**
canned	OLTP/BI	BI
ad hoc	BI	BI

Table 1: Characterization of workloads

The horizontal axis indicates whether queries are issued *interactively* or as part of *batch* jobs. In the case of interactive invocation, the user is waiting for the results of each query before submitting the next one. A subsequent query may depend on the result of its predecessor, so even long-term monitoring of query patterns will not necessarily yield a good prediction model for query sequences. The opposite is true for batch workloads, where all queries are known in advance.

OLTP workloads are interactive, canned workloads that typically consist of a large number of small uniformly-sized queries. Results must be returned quickly for the sake of a good user experience. BI workloads, on the other hand, may contain queries from all quadrants of Table 1: interactive OLTP-like queries may be interleaved with long-running canned batch workloads that create business reports or statistics. BI workloads additionally include ad hoc queries, e. g., if a business analyst interactively performs drill-down analysis or requests a custom report to be run as an overnight batch job.

3 Service Level Objectives

From a user's point of view, a database system performs well if the performance requirements the user cares about are met. A first prerequisite is to translate user-defined performance requirements into a common set of metrics that can be obtained through monitoring. Examples of such metrics include *execution time* (the elapsed wall clock time between the start and completion of a query), *throughput* (number of successfully executed queries in a given time span), and *CPU time* (time the CPU is available for a specific query).

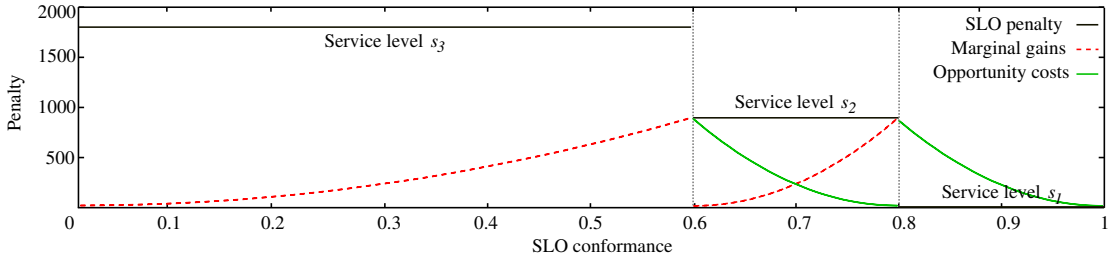

Figure 1: Visualization of SLO constraint d

A service level objective (SLO) is formed by a combination of one or more performance goals and an associated priority, which typically depends on the penalties incurred if the goals are not met. Often, these objectives do not apply to all queries, but instead must be satisfied for a certain percentage. Such *SLO conformance* metrics are defined as ratio of the number of queries that meet an SLO goal to the total number of queries.

The SLOs for the workload categories described in Section 2 differ in the way performance is measured - either in terms of individual queries or in terms of groups of queries. Users are explicitly aware of the individual response times of interactively submitted queries, but the response time for batch jobs is measured for a set of queries as a whole. Similarly, performance for canned queries (e. g., a canned report or OLTP query) tends to be measured and reported in terms of the query class as a whole, whereas ad hoc queries are measured individually.

An example for an SLO in the OLTP context is the so-called *step-wise SLO* that consists of one or more *percentile constraints*. Since users typically expect fast responses for OLTP queries, percentile constraints require $n\%$ of all service requests to be processed within x seconds. Otherwise, a penalty p for every m percentage points under fulfillment is due. A percentile constraint implicitly defines an SLO penalty function with n steps (service levels). The penalty function for a constraint d (90% in $< 5s$; $p = \$900$ per 20 percentage points, maximum penalty $\$1800$) is shown in Figure 1 (black solid lines).

In contrast, batch workloads are typically deadline-driven and may incur penalties if work is not completed before a given deadline. This translates directly into an execution time constraint. Another common SLO for batch workloads specifies a lower-bound for the throughput, i. e., the batch workload does not have a fixed deadline, but should be assigned a specific portion of the available system resources.

4 Workload Management

The goal of workload management for database systems is to increase user satisfaction by meeting SLOs. Note that in general, neither the customer nor the provider benefits from over-fulfilled SLOs, because there is no advantage to providing results before a given deadline, and the execution time requirements already ensure a responsive interaction with the DBMS. Over-fulfilling an SLO will not excuse a future SLO violation and moreover could raise unreasonable performance expectations.

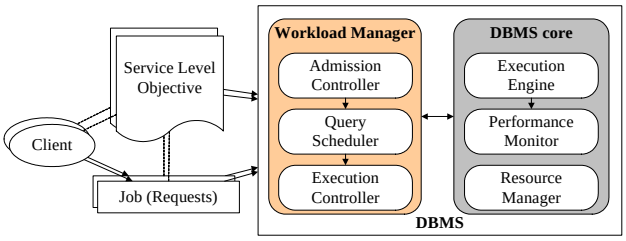

Figure 2: Generic workload management architecture

Figure 2 sketches a generic workload management architecture. The DBMS core offers the following components: the *Execution Engine*, which manages the processing of queries, the *Resource Manager*, which provides priority based allocation of resources to queries, and the *Performance Monitor* for monitoring the execution of queries. Modern DBMS offer several knobs for tuning workload performance at all points in the workload life cycle. Workload management begins with the opportunity to prevent a new query from being placed on an execution queue, continues with queuing and

scheduling decisions, and includes the ability to control the execution of a running query. These mechanisms are implemented by the *Admission Controller*, the *Query Scheduler*, and the *Execution Controller*, respectively. The latter contains a rule base for identifying unexpected overload situations and deciding which workload management action should be performed for which queries. The workload management decisions are driven by the SLOs that are annotated to each batch job and interactive query of every client. The objectives must be made available for the DBMS prior to the execution of the job.

Admission control can prevent potential problem queries from being started in the first place. Query scheduling optimizes the order of execution and the number of concurrently running queries by deciding when to admit which query. Admission control and scheduling operate atop of the database layer, and can be implemented without modifying the database core engine. If the DBMS offers interfaces to control already running queries, then finer grained control of request execution is possible. The presence of execution control mechanisms that can, e. g., kill or suspend and resume queries at run-time can further improve performance. For example, killing a query that hogs system resources for an unexpectedly long time can limit the negative impact on the overall execution performance. Similarly, if the DBMS offers prioritization mechanisms for allocating resources like memory, CPU, and locks, then complex queries can be adjusted to lower priorities when necessary, leaving enough resources for newly arriving interactive queries with higher priorities.

5 Related Work

Workload Management Techniques Regarding work on workload management techniques for resource allocation, we share a focus with researchers such as [1, 6, 15, 21], who consider how to govern resource allocation for queries with widely varying resource requirements. For example, Davison and Graefe [6] present a framework for resource allocation based on concepts from microeconomics. Their framework aims at reducing response time of queries in a multi-user environment. The central component is a resource broker that assigns operators the share of the resource they are willing to pay for. Weikum et al. [24] discuss what metrics are appropriate for identifying the root causes of performance problems in an OLTP workload (e. g., overload caused by excessive lock conflicts). They focus on tuning decisions at different stages such as system configuration, database configuration, and application tuning.

Query progress indicators (e. g., [3, 14]) attempt to estimate a running query's degree of completion. Research in this area is complementary to our goals, and potentially offers a means to identify long-running queries at early stages – before the workload has been negatively impacted.

Recently, research has been done on query suspension and resumption, e. g., [2, 4]. When a query is suspended, the DBMS releases all resources held by the query. At a later point in time, the query can be resumed, ideally without wasting a large amount of work that has been done prior to the suspension. We believe that database products will implement these techniques in the near future.

Workload Management Implementations Some workload management techniques for admission control, scheduling, and execution management policies are already implemented in products such as those by HP (e. g., Workload Manager for Neoview [8]), IBM (e. g., Query Patroller for DB2 [17], Optimization Service Center [9], zSeries [13]), Microsoft (SQL Server [16]), Oracle (Discoverer [19], Resource Manager [20]), and Teradata (Workload Manager [23]). We only present a short overview, a more detailed description can be found in [12].

Admission control uses thresholds to prevent overly-expensive queries from running in the system. Database vendors provide different metrics like optimizer costs or processing time estimates. Queries for which the values of the metrics exceed administrator-defined thresholds are either rejected or put on hold. The latter case requires the administrator to decide whether to submit the query to the database or to reject it.

Some databases allow the administrator to limit the level of concurrency on the database system and to schedule the delayed queries. There are three approaches for limiting the concurrency: limit the maximum resource utilization, restrict access to database objects like tables, indexes, and views, and limit the number of

simultaneous queries in the database system. Delayed queries are typically managed in three different types of scheduling queues: FIFO, size-based, and priority-based. Size-based queues prevent short-running queries from getting stuck behind long-running queries, thus enforcing a consistent elapsed time for short queries in the presence of long-running queries. Priority-based queues enforce the preferred execution of high-priority queries compared to their lower-priority counterparts. It is the task of the database administrator to define priorities for the users in order to balance the performance of the system. Queries are then ordered in the queue according to the priority of the submitting user.

Execution control is usually implemented by using rules where a condition triggers an execution management action. The different vendors support a variety of metrics to be used in the conditions, e. g., cardinality, CPU time, number of I/Os, and elapsed wall clock time. Almost all database vendors implement some notification mechanism to inform the administrator about exceptional situations, e. g., when the elapsed time of a query exceeds a specified threshold. It is then the task of the administrator to analyze and tackle workload management problems. If the administrator does not take action, the query runs to completion. In addition to that, HP's and Teradata's Workload Manager can be configured to automatically kill a query.

6 Workload Management for OLTP Workloads

This section focuses on OLTP-style workloads that consist of a multitude of a priori unknown short-running transactions that may be started by clients at any time. Each transaction consists of a set of interactively submitted queries for which the user expects short response times. Therefore, users often negotiate SLOs similar to the step-wise SLOs introduced in Section 3, which limit the response time for a percentage of transactions. From the workload management perspective, the main challenge is to apply a query scheduling policy that meets the SLO requirements for as many users as possible. A common approach in such settings is to provide priority-based queues to manage pending queries. Typically, the priority of queries is defined externally, e. g., by a database administrator and usually depends on the priority of the respective user. If only a percentage of all queries must meet the performance requirements, static prioritization tends to over-fulfill SLOs of high-priority users because their queries are almost always processed in time at the expense of their lower-priority counterparts.

Therefore, our approach derives an adaptive penalty based on an economic model and annotates the queries with the penalty information. These penalties are used to optimize the execution order of the pending queries. We define the penalty of a query as the maximum of two economic cost functions. *Opportunity costs* (monotonically decreasing parts of the parabolas in Figure 1) model the danger of falling to the next lower service level. If the current SLO conformance converges to the next lower service level, the penalty for processing the service too late increases because delaying an additional query increases the likelihood of an ultimate SLO violation. *Marginal gains* (monotonically increasing parts of the parabolas in Figure 1) model the chance that a service class re-achieves a higher service level. If this appears to be "within reach", individual queries are increasingly penalized until eventually the higher level is reached.

The computation of penalties, scheduling algorithms for pending queries, and a performance analysis are described in [7, 10]. The experimental results show the effectiveness of our novel adaptive penalization approach, which provides a higher overall SLO conformance by reducing the over-fulfillment for high-priority clients.

7 Workload Management for BI Workloads

BI workloads contain a wide variety of requests, ranging from batch jobs to short-running interactive queries to ad hoc queries of varying complexity. For the batch part of the workload, we focus here on deadline-driven SLOs. The optimization goal for these jobs is to minimize the time needed to complete the workload. Interactively submitted queries, on the other hand, require short response times, because users easily become dissatisfied if they must wait for responses too long. SLOs for interactive queries therefore require an execution time below a

given threshold. If a batch and an interactive job are running simultaneously on the database, the challenge is to optimize the execution of all jobs subject to their SLOs. Additionally, a workload management system must be capable of dealing with ad hoc designed queries that may have unknown execution characteristics or may cause performance problems.

Admission Control One approach for optimizing the performance of BI workloads is to reject overly-expensive queries that may hog system resources and thus prevent concurrently running queries from making progress. The administrator may choose to run these queries in a controlled manner, e. g., in isolation, to minimize the impact of these problem queries on others. The challenges to be addressed are twofold: First, a threshold for identifying overly-expensive queries must be found. If the value is set too low, many queries will be rejected. A threshold that is too high may admit too many expensive queries, resulting in exceptional situations at run-time. Second, admission control requires accurate optimizer estimates and knowledge about how a query impacts concurrent queries. Query optimizers do a good job estimating the costs of queries when requirements like uniformity of data, independence of attributes, and up-to-date statistics are met. However, in the BI context, data may be heavily skewed and statistics cannot be kept up-to-date for update-intensive workloads. Therefore, the optimizer might return estimates that are off by orders of magnitude from the actual processing costs, making the estimates unusable for admission control.

Query Scheduling The task of the query scheduling component is to decide when to admit which query. Scheduling must obey inter-query dependencies, i. e., some requests cannot be reordered arbitrarily because they depend on the results of other requests. Another challenge is to determine the optimal number of concurrently running queries in the database that would maximize the usage of all system resources. The optimal number of queries in the system depends on parameters like the database configuration, the underlying hardware, and the requests themselves, and might even vary during the execution of the workload. In practice, finding an optimal solution to the scheduling problem itself is not feasible because of the high complexity and the large instance sizes containing hundreds of requests. Therefore good heuristics must be found. Additionally, a reasonable reordering of workload requests needs to consider the impact requests have on one another. Some requests are potentially working well together while others interfere with each other.

A prerequisite for scheduling is a concise representation of benefits and detriments of concurrent request execution. This information can be subsumed in a "synergy matrix" as exemplified in Figure 3. Each entry (R_i, R_j) in the two-dimensional matrix is a numerical value denoting the relative impact of the parallel execution of requests R_i and R_j. There exist several metrics for quantifying the (dis)advantages, like consumed CPU cycles, disk accesses, or execution time. In this work, we focus on the ratio of the elapsed times measured for parallel and sequential execution of requests. A value less than 1 indicates that the parallel execution is faster than the sequential execution. For example, the synergy value 0.70 for R_1 and R_2 (light gray) denotes that executing the two requests concurrently takes 70% of the time it takes to execute them sequentially. In contrast, the parallel execution of R_3 and R_4 takes longer than running them in sequence (dark gray). An empty entry ("—") in the matrix indicates that a synergy value is not yet available.

	R₁	**R₂**	**R₃**
R₂	0.70		
R₃	-	0.85	
R₄	-	0.92	1.10

Figure 3: Synergy matrix

In order to increase the quality of the scheduling results, the matrix must be populated with as many values as possible. This can be accomplished through either analysis or monitoring. The former approach applies a white box technique and is based on an analysis of the workload's requests in order to determine potential synergies before the actual execution. Sources of such synergies stem from caching behavior and multi-query optimization (MQO). In the context of MQO, extensive research has been done on identification of common sub-expressions [5, 22] and cooperative scans [25]. Drawbacks of the white box approach are that some of the required information may not be available prior to the execution of a request and that predictions errors in the analysis phase may result in incorrect assumptions about individual requests and, thus, potential synergies.

The monitoring approach treats the workload's jobs as black boxes, i. e., does not make any assumptions about their characteristics. It monitors the execution of the requests and iteratively derives information about potential synergies. For example, O'Gorman et al. [18] employ this approach by running all pairs of TPC-H requests both concurrently and sequentially and then comparing the number of disk accesses. A substantial drawback of the black box approach is that the synergy matrix is only populated during workload execution. Another difficulty is to infer (dis)advantages for pairs of queries if monitored data is only available for a whole set of concurrently executing queries.

Analysis and monitoring are complementary approaches and can be combined for better results. Prior to the execution of the batch, analysis can be used to provide an initial population of the matrix, while monitoring during run-time can be used to refine inaccurate results from the analysis and provide values for requests that cannot be analyzed or that interfere with each other unexpectedly.

Execution Control There are two major challenges for run-time execution control of queries. First, execution control needs to detect exceptional situations based on the metrics that can be monitored at run-time. Identifying a problem could be as easy as comparing the actual elapsed time of a query to a threshold provided by, e. g., the user or the administrator. More sophisticated conditions for triggering an execution control action could include additional metrics like CPU time and number of disk I/Os. Although a greater number of metrics provides a higher flexibility, monitoring the metrics may cause overhead at run-time, thus slowing down the processing of the requests. Even if a set of metrics has been identified, the challenge is to set thresholds. Practitioners with experience in workload management can attest not only the importance of good thresholds but also the difficulty of finding these values. Second, the execution management needs to choose from a set of corrective actions like killing or suspending a query. If a query is killed, the execution control needs to decide when, if at all, to resubmit it. Similarly, an appropriate policy for resuming a suspended query must be found. Of course, the execution control must obey the service level objectives, e. g., high-priority queries with tight deadlines might not be killed, even if they hog the system resources for a long time.

Experimental Framework for Workload Management In order to provide a more application-oriented approach for workload management, we have developed an experimental framework, introduced in [11] for evaluating the effectiveness of workload management techniques. The architecture of the framework follows the architecture in Figure 2. Admission Controller, Query Scheduler, and Execution Controller represent knobs that can be adjusted to select from a variety of workload management policies and algorithms. Our framework is not limited to workload management policies already implemented by existing database systems and tools, but allows us to experiment with new workload management concepts. Furthermore, we implemented a simulator for the execution engine, which mimics the execution of a workload in a database system. We model a workload as composed of one or more jobs. Each job consists of an ordered set of typed queries and is associated with a performance objective. Each query type maps to a tree of operators, and each operator in a tree maps in turn to its resource costs. Our current implementation associates the cost of each operator with the dominant resource associated with that particular operator type (e. g., disk or memory). The life cycle of the data that drives the experimental runs is described in [11].

8 Summary

In this paper, we have characterized OLTP and BI workloads and identified factors in workload generation and submission that impact their service level objectives (SLOs). We outlined SLOs in the database context and the current state of the art in workload management techniques for enforcing these objectives. We summarized our contributions for managing OLTP workloads by adaptively penalizing individual queries. We looked at BI workload management where we sketched at which points in a workload's life cycle management is applicable, and presented a synergy matrix that characterizes the impact of running particular batch queries concurrently.

Finally, we overviewed our experimental framework for testing the impact of the various workload management techniques on the execution of workloads. For more details about this work, we refer readers to [7, 10, 11, 12].

References

[1] M. J. Carey, M. Livny, and H. Lu. Dynamic Task Allocation In A Distributed Database System. In *Proc. of the* 5^{th} *Intl. Conf. on Distributed Computing Systems (ICDCS)*, pages 282–291, 1985.

[2] B. Chandramouli, C. N. Bond, S. Babu, and J. Yang. Query Suspend And Resume. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, 2007.

[3] S. Chaudhuri, R. Kaushik, and R. Ramamurthy. When Can We Trust Progress Estimators For SQL Queries? In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 575–586, 2005.

[4] S. Chaudhuri, R. Kaushik, R. Ramamurthy, and A. Pol. Stop-and-Restart Style Execution for Long Running Decision Support Queries. In *Proc. of the* 33^{rd} *Intl. Conf. on Very Large Data Bases (VLDB)*, 2007.

[5] S. R. Choenni, M. L. Kersten, and J. F. P. van den Akker. A Framework for Multi-query Optimization. In *Proc. of the Intl. Conf. on Management of Data (COMAD)*, 1997.

[6] D. L. Davison and G. Graefe. Dynamic Resource Brokering for Multi-User Query Execution. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 281–292, 1995.

[7] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, and A. Kemper. Adaptive Quality of Service Management for Enterprise Services. *Accepted for publication in ACM Transactions on the Web (TWEB)*, 1, 2008.

[8] HP NeoView Workload Management Services Guide, August 2007.

[9] IBM Optimization Service Center for DB2 for z/OS. `http://www-306.ibm.com/software/data/db2/zos/downloads/osc.html`.

[10] S. Krompass, D. Gmach, A. Scholz, S. Seltzsam, and A. Kemper. Quality of Service Enabled Database Applications. In *Proc. of the* 4^{th} *Intl. Conf. on Service-Oriented Computing (ICSOC)*, pages 215–226, 2006.

[11] S. Krompass, H. Kuno, U. Dayal, and A. Kemper. Dynamic Workload Management for Very Large Data Warehouses: Juggling Feathers and Bowling Balls. In *Proc. of the* 33^{rd} *Intl. Conf. on Very Large Databases (VLDB)*, pages 1105–1115, 2007.

[12] S. Krompass, H. Kuno, J. Wiener, U. Dayal, and A. Kemper. Managing Long-Running BI Queries: To Kill Or Not To Kill, That Is The Question, 2008. Submitted for publication; please contact authors for full version of paper.

[13] N. Lei. Workload Management for DB2 Data Warehouse. `http://www.redbooks.ibm.com/redpapers/pdfs/redp3927.pdf`.

[14] G. Luo, J. F. Naughton, and P. S. Yu. Multi-query SQL Progress Indicators. In 10^{th} *Intl. Conf. on Extending Database Technology (EDBT)*, pages 921–941, 2006.

[15] M. Mehta and D. J. DeWitt. Dynamic Memory Allocation for Multiple-Query Workload. In *Proc. of the Nineteenth Intl. Conf. on Very Large Data Bases*, 1993.

[16] Microsoft SQL Server 2005 Books Online. `http://msdn2.microsoft.com/en-us/library/ms190419.aspx`, September 2007.

[17] B. Niu, P. Martin, W. Powley, R. Horman, and P. Bird. Workload Adaptation In Autonomic DBMSs. In *CASCON '06: Proc. of the 2006 Conf. of the Center for Advanced Studies on Collaborative Research*, 2006.

[18] K. O'Gorman, D. Agrawal, and A. E. Abbadi. Multiple Query Optimization by Cache-aware Middleware Using Query Teamwork. *Softw. Pract. Exper.*, 35(4):361–391, 2005.

[19] Oracle Discoverer Administrator Administration Guide 10g (9.0.4). `http://download.oracle.com/docs/html/B10270_01/adpqta01.htm`.

[20] A. Rhee, S. Chatterjee, and T. Lahiri. The Oracle Database Resource Manager: Scheduling CPU Resources at the Application Level. `http://research.microsoft.com/~jamesrh/hpts2001/submissions/`, 2001.

[21] B. Schroeder, M. Harchol-Balter, A. Iyengar, and E. M. Nahum. Achieving Class-Based QoS for Transactional Workloads. In *Proc. of the* 22^{nd} *Intl. Conf. on Data Engineering (ICDE)*, page 153, 2006.

[22] S. N. Subramanian and S. Venkataraman. Cost-based Optimization of Decision Support Queries Using Transient-views. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 319–330, 1998.

[23] Teradata. Teradata Dynamic Workload Manager User Guide, September 2006.

[24] G. Weikum, C. Hasse, A. Mönkeberg, and P. Zabback. The COMFORT Automatic Tuning Project. *Information Systems*, 19(5):381–432, 1994.

[25] M. Zukowski, S. Heman, N. Nes, and P. Boncz. Cooperative Scans: Dynamic Bandwidth Sharing in a DBMS. In *Proc. of the* 33^{rd} *Intl. Conf. on Very Large Databases (VLDB)*, pages 723–734, 2007.