Workload Management in Database Management Systems: A Taxonomy

Mingyi Zhang, Patrick Martin, Wendy Powley, Jianjun Chen

Abstract—Workload management is the discipline of effectively monitoring, managing and controlling work flow across computing systems. In particular, workload management in database management systems (DBMSs) is the process or act of monitoring and controlling work (i.e., requests) executing on a database system in order to make efficient use of system resources in addition to achieving any performance objectives assigned to that work. In the past decade, workload management studies and practice have made considerable progress in both academia and industry. New techniques have been proposed by researchers, and new features of workload management facilities have been implemented in most commercial database products. In this paper, we provide a systematic study of workload management in today's DBMSs by developing a taxonomy of workload management techniques. We apply the taxonomy to evaluate and classify existing workload management techniques implemented in the commercial databases and available in the recent research literature. We also introduce the underlying principles of today's workload management technology for DBMSs, discuss open problems and outline some research opportunities in this research area.

Index Terms—Taxonomy, Workload Management, Database Management Systems

1 Introduction

A database workload is a set of requests that have some common characteristics such as application, source of request, type of query, business priority, and/or performance objectives [72]. For both strategic and financial reasons, some business organizations are consolidating multiple individual database servers onto a shared database server to serve as the single source of corporate data [3] [9]. This trend of database server consolidation means that multiple types of workloads are simultaneously present on a single database server. These workloads may include on-line transaction processing (OLTP), which consists of short and efficient transactions that may require only milliseconds of CPU time and very small amounts of disk I/O to complete, as well as Business Intelligence (BI) workloads [31], which include longer, more complex and resource-intensive queries that can require hours or an even longer time to complete. Workloads submitted by different applications or initiated from distinct business units may have unique performance objectives (goals) that need to be strictly satisfied. The performance objectives of a workload are normally derived from a formal service level agreement (SLA).

On a shared database server, there is an interdependence among the concurrently running workloads that results from the workload's competition for the shared system resources, such as system CPU, main memory, disk I/O, network bandwidth and various queues. If a workload, *e.g.*, an operational BI workload, is allowed to consume a large amount of shared system resources without any control, the concurrently running workloads may have to wait for the workload to complete and to release its used resources, thereby resulting in waiting workloads missing their performance goals and the entire database server suffering degradation in performance. As workload requests present on a database server can fluctuate rapidly among multiple types, it becomes impossible for database administrators (DBAs) to manually adjust the system configurations in order to maintain the workload's performance requirements during their run time. Thus, workload management becomes necessary and critical to effectively control the processes of different types of workloads and manage shared system resources to achieve a set of per-workload performance goals in a complex workload mix environment.

Workload management is the discipline of effectively monitoring, managing and controlling work flow across computing systems [8] [74]. In particular, workload management for database management systems (DBMSs) is the process or act of monitoring and controlling work (or requests) executing on a database system in order to make efficient use of system resources in addition to achieving any performance objectives assigned to that work [3]. Thus, the primary goals of workload management for a DBMS are: 1) to maintain the DBMS running in an optimal state, *i.e.*, neither under-loaded nor over-loaded, 2) to ensure that all workloads meet their desired performance objectives (if any), and 3) to balance resource

- *Mingyi Zhang and Jianjun Chen are with Huawei America Research, Santa Clara, California, USA 95050. E-mail: mingyi.zhang1@ huawei.com, jianjun.chen1@huawei.com.*
- *Patrick Martin and Wendy Powley are with School of Computing, Queen's University, Kingston, Ontario, Canada K7L3N6. E-mail: martin@cs.queensu.ca, wendy@cs.queensu.ca.*

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2767044, IEEE Transactions on Knowledge and Data Engineering

demands of workloads running concurrently and maximize performance of the entire system. Niu *et al*. [59] observe that the specific goal of workload management is to address the conflict of cost sharing and SLA satisfaction. Cost sharing aims to consolidate more user applications onto a single database server in order to minimize business cost and maximize its investment return. While, on the other hand, the goal of SLA satisfaction is to achieve performance requirements of all work running concurrently on the single database server. It is easy to observe the conflict between these two perspectives as better cost sharing (more users sharing a database server) can lead to poorer SLA satisfaction (resulting in significant resource competition). To resolve this issue, researchers and engineers [9] [37] [47] believe that instead of trying to develop a single sophisticated workload management technique, a workload management system that employs multiple types of workload management techniques becomes necessary.

The primary objective of this paper is to provide a systematic study of workload management in today's DBMSs by surveying the workload management systems and techniques implemented in major commercial databases as well as those proposed in the research literature. In a previous survey of workload management for DBMSs, Niu *et al.* [59] proposed a general framework of autonomic workload management for DBMSs and used a set of criteria to evaluate the existing facilities for supporting the features of autonomic workload management in DBMSs. In another overview of workload management in DBMSs, Aboulnaga *et al.* [1] discussed and compared a set of workload management techniques used in the data warehouse and MapReduce systems. While, Krompass *et al.* [37] [39] presented a process of workload management in DBMSs and examined various control policies at each control point in the workload management process. In this study, we propose a taxonomy of workload management techniques to classify workload management techniques and evaluate the state of the art in today's workload management for DBMSs.

The remainder of the paper is organized as follows. Section 2 reviews the background of the current workload management technology for DBMSs and introduces the underlying principles. Section 3 presents a taxonomy of workload management techniques developed based on the main functions presented in a workload management process and the typical techniques presented in current studies and practice. We survey typical workload management systems and techniques and use the taxonomy to classify and evaluate the systems and techniques in Section 4. Finally, we summarize the contents of this paper, based on the examination of the progress made on workload management, discuss open problems, and suggest future research in Section 5.

2 Background

Research in workload management for DBMSs has been mainly driven by commercial database vendors. As databases are becoming increasingly large and complex, providing features of workload management in the DBMSs to help the systems to achieve desired performance objectives has been a key factor for business success. By surveying the workload management facilities provided in today's commercial DBMSs and techniques proposed in the recent research literature, we present background information on workload management technology for DBMSs and introduce the underlying principles in this section.

In the use of the workload management facilities in commercial databases, *e.g.*, IBM DB2 Workload Manager [30], Microsoft SQL Server Resource and Query Governor [50] [51], Teradata Active System Management [71] [72], Oracle Database Resource Manager [61], and Pivotal Greenplum Databases [63], three major steps are suggested to effectively manage the wide variety of work executing concurrently in a database server:

- Explicitly understand performance objectives (or goals) of all requests based on a formal SLA (if any);
- Clearly identify arriving requests in the database server;
- Impose appropriate controls on the requests to manage their behaviors so that the requests can make steady progress towards the desired performance objectives;

These principles are not only applied to the workloads on traditional DBMSs, but are also able to extend to other workloads, such as the real-time analytical workload on in-memory databases or tables [52] [53] [62] [68]. In the following subsections, we discuss each of these principles of workload management in detail.

2.1 Performance Objectives and Management Policies

As introduced previously, a request executing in a database server may have an assigned business priority (or a business-importance level) and performance objective, and both are defined in terms of a SLA. The mapping from the business-level SLA to the specific business priorities and performance objectives can be a non-functional process (that is, a function cannot be defined for the mapping) that requires business mapping rules along with knowledge shared by the DBAs. The performance objectives can be expressed by one or more performance metrics.

Typical performance metrics include *response time*, the elapsed time between the start and completion of a request, *throughput*, the number of requests completed in a time unit, and *request execution velocity*, the execution speed of a request in a database system [74]. Request execution velocity can be simply described as the ratio of the expected execution time of a request to the actual time the request spent in the system, *i.e.*, the total time of execution and delay, where the expected execution time can be obtained from historical observations in the system's steady state. If an execution velocity is close to 1, the delay of the request is small, while, an execution velocity close to 0 indicating a significant delay. In particular, high priority requests, such as the ones that directly generate

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2767044, IEEE Transactions on Knowledge and Data Engineering

revenue for business organizations, or those issued by a CEO or VP of the organizations, may expect a quick response, and thus they need to complete quickly. Low priority requests, such as the ones for generating routine business reports, can wait longer to get results. In using request execution velocity, performance objectives as well as business priorities of requests can be captured. That is, by checking if a request's execution velocity is close to 1, it can be known that the request (no matter a low or high priority) has met its desired performance objective or not.

To efficiently manage an end user's work, requests are grouped into workloads. As a result, performance objectives of the requests can be expressed relative to a workload. In many situations, workload performance objectives are described in averages or percentiles, such as the average response time of transactions in an OLTP workload, or, x% queries in a workload for data warehousing decision support system complete in y time units or less [9]. A workload with a high business priority may be associated with a high performance requirement, or, larger x and smaller y values. For non-goal request workloads, there are typically no associated performance objectives. However, their performance may be inferred from a system resource allocation plan, such as "non-goal workloads may consume no more than r% of the total processor resources" [9].

In a workload management process, workload management policies are important in controlling the behavior of running requests, where policies are the plans of an organization to achieve its objectives [44] [57]. Workload management policies may include determining if a request can be admitted to a database system, how long the request has to wait in a queue for available shared system resources, and how fast the request can be executed in the database system. The policies are typically derived from the defined workload business priorities and performance objectives, and the policies may be applied to all points in a workload management process. At different points, policies may have different specifications and rules, such as admission policies used for specifying how a request would be controlled during its admission to the system, scheduling policies used for guiding the request scheduling processes of workload management facilities, and execution control policies used for defining dynamic execution control actions [37] [47] [80].

2.2 Request Identification

Having explicitly defined performance objectives and management policies to manage the end user's work to achieve the desired performance objectives, current practice shows that arriving requests need to be clearly identified when they present on a database system so that the queries can be properly managed [9] [30]. Thus, various workload definition approaches are used to identify the incoming requests.

The term *workload* is also used to refer to an object in today's commercial databases, which is defined for monitoring and controlling a set of requests [30] [72]. Workload definition approaches use classifications to map arriving requests to workloads. A defined workload can be assigned a priority (at the business level) based on the SLA specifications when the workload is defined. A business transaction processing workload, such as data insertions generated by cashiers in a store, is always assigned high business priority as the transactions directly generate revenue and should complete promptly. On the other hand, a business analysis workload, such as a set of business report generation queries for the store (*i.e.*, a report-generation batch workload) may be assigned a lower priority as report generation is a daily routine and may be done in any idle time window during the day. High business priority workloads require high-priority access to shared system resources so that their performance objectives can be reached and guaranteed. Grouping requests into workloads simplifies the task of allocating resources and provides visibility into how system resources are being used by each workload [30] [50] [61] [72].

Assigning a request to a workload can be done based on the request's operational properties, such as origin or type [30] [72], or by applying a set of user-defined criteria functions [50]. A request's origin, which indicates "who" is making the request, can include properties, such as the application name, user name, application's session ID and client IP address. A request's type specifies "what" the characteristics of the request are, such as types of statements, estimated costs or estimated cardinalities. The types of request statements may include READ, WRITE, Data Manipulation Language (DML), and Data Definition Language (DDL) [67]. Estimated costs or cardinalities predict the consumption of various shared system resources. Criteria functions are typically scalar functions that contain the logic to classify the incoming requests into workloads, where the logic can be derived from request properties [50].

TABLE 1
THREE TYPES OF CONTROLS IN A WORKLOAD MANAGEMENT PROCESS

Control Type	Description	Control Point	Associated Policy
Admission Control	Determines whether or not an arriving request can be admitted into a database system	Upon arrival in the database system	Admission control policies derived from a workload management policy
Scheduling	Determines the execution order of requests in batch workloads or in wait queues	Prior to sending requests to the database execution engine	Scheduling policies derived from a workload management policy
Execution Control	Manages the execution of running requests to reduce their performance impact on the other requests running concurrently	During execution of the requests	Execution control policies derived from a workload management policy

2.3 Workload Control

Current research proposes that a workload management process in DBMSs may involve three different types of controls, namely, *admission*, *scheduling* and *execution control* [37] [39] [80], as listed in Table 1, and the controls are

guided by specified workload management policies [38] [80]. Admission control determines whether or not newly arriving requests can be admitted into a database system, thereby avoiding an increase in load while the system is busy. It identifies "problematic" requests, such as the long-running and resource-intensive requests, and makes appropriate admission decisions. The decision is based on the estimated costs of the arriving requests and the specified workload management policies (specifically admission control policies). The cost is typically estimated by the database query optimizer [13] [25]. If a request's estimated cost exceeds the pre-defined admission threshold, the request may be queued for later admission or rejected with some returned message. The pre-defined admission thresholds are determined based on the admission control policies. The thresholds can include the upper limits for the estimated resource usage of a request, the estimated execution time of the request, and the number of requests running concurrently (the multi-programming levels) in the database system. Workloads with different priorities can be associated with different admission control policies, and therefore have different sets of threshold values. A high priority workload usually has higher (less restrictive) thresholds, so high priority requests can be guaranteed to be admitted into the database system for execution.

Request scheduling determines the execution order of requests in batch workloads or admitted requests in wait queues (*e.g.*, priority queues) and decides when the requests can be sent to the database execution engine for execution based on the workload management policies (specifically scheduling policies). The challenge of request scheduling is to determine the optimal number of requests with various characteristics, priorities, and resource demands that can run concurrently in a database system while maintaining the system in an optimal state and meeting the performance objectives for all workloads. Traditionally, the multi-programming levels (MPLs), a database system's threshold-based configuration parameter, are used to manage the system load. MPLs specify the upper limit of the number of queries that are allowed to run concurrently in a database system. If the MPL value is too large, the system can become over-utilized, while, on the other hand, if the MPL value is too low, the system may be under-utilized. In both cases, system performance suffers. For the same database system, different types of workloads have different optimal MPLs. Request scheduling aims to dynamically set MPLs for each of the workloads to decide which and how many requests can be sent to the database to execute concurrently based on a specified scheduling policy.

In contrast with the *admission control* and *scheduling*, which are applied to requests before their execution, the *execution control* is imposed on a request during the run time. The main goal of execution control is to dynamically manage a running request in order to limit its impact on other running requests, *e.g.*, by slowing down the request's execution speed and freeing up shared system resources for use by the other requests. Since query costs estimated by the database query optimizer may be inaccurate, long-running and resource-intensive queries may get the chance to enter a system while the system is experiencing a high load. These "problematic" requests compete with others for the limited available system resources and result in the requests obtaining insufficient resources and missing their desired performance objectives. Execution control manages the running of the problematic requests based on an execution control policy and determines to what degree the control should be applied.

3 A Taxonomy of Workload Management Techniques

In the past decade, considerable progress has been made in workload management for DBMSs. New features of workload management facilities have been implemented in commercial DBMSs and new techniques have been proposed by researchers. However, the descriptions of the facilities and techniques in publically available documentation are very different in terms of their terminology, even though their primary goals are the same, namely to achieve a set of per-workload SLAs in a complex mixed workload environment. To facilitate the study and understand the state of the art of the current workload management technology for DBMSs, we develop a taxonomy shown in Figure 1 to categorize workload management techniques based on the main features of the techniques. The purpose of the taxonomy is to:

- Classify the typical workload management techniques proposed in the research literature and used in workload management facilities provided in DBMSs;
- Highlight the relative strengths and weaknesses of existing techniques and to point out deficiencies in the current set of techniques;

Our taxonomy is developed based on the controls involved in the workload management process discussed previously, and the techniques currently suggested by commercial database vendors and in the recent research literature. We categorize the workload management techniques for DBMSs into four major classes, namely, *workload characterization*, *admission control*, *scheduling* and *execution control*. Within a class, the workload management techniques are further divided into subclasses based on their distinct mechanisms. In this section, we discuss the main features of typical techniques in each class. These features are used to classify a particular workload management approach.

3.1 Workload Characterization

Workload characterization is essential for a workload management process as it provides the fundamental information about a workload to its controllers. Workload characterization can be described as the process of identifying characteristic classes of a workload in the context of its properties, such as costs, resource demands, business priorities, and/or performance requirements. A business transaction processing workload, for instance, is often characterized as having low cost, few resource demands, high business priority, and requiring good performance.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2767044, IEEE Transactions on Knowledge and Data Engineering

While, on the other hand, a business analysis workload can be characterized as having high cost, large resource demands, low business priority, and requiring best-effort (implicit) performance objectives. The *workload definition* discussed in Section 2 is considered as a process of workload characterization as, when a workload is defined, it is also characterized with regard to its assigned business priorities, estimated costs and expected performance behaviors.

We divide workload characterization techniques into two types, namely, *static characterization* and *dynamic characterization*, as shown in Figure 1. Static workload characterization defines the workloads before requests arrive and allocates shared system resources to the defined workloads. This type of technique is widely employed in workload management facilities provided in commercial DBMSs [30] [50] [72]. The main features of the techniques are the differentiation of arriving requests based on their operational properties discussed in Section 2, the mapping of the requests to a workload, and the resource allocation to the workloads for their execution. Resource allocation is typically done based on the priority assigned to a workload, such as high, medium or low. A workload with higher business priority would have a higher priority to access shared system resources. Once assigned to a priority level (at the system level), a workload has the rights defined for the priority level to access shared system resources. However, the priority may be dynamically changed during the workload execution based on the workload's performance requirements and actual performance behavior [9], as explained in the discussion of Execution Control in the following sections.

Dynamic workload characterization identifies the type of a workload when it is present on a database server (*e.g.*, an online transaction processing or an online analytical processing workload). Typical techniques proposed in the research literature for workload classification is machine-learning [19] [73]. In using this technique, the system learns the characteristics of sample workloads running on a database server, builds a workload classifier and uses the workload classifier to dynamically identify unknown arriving workloads on the database server.

3.2 Admission Control

Traditionally, admission control in OLTP systems ensures that the number of client connections is kept below a threshold so that the resource contention level among concurrent requests is controlled. In the system, if the number of requests increases, throughput of the system increases up to some maximum. Beyond the maximum, it begins to decrease dramatically as the system starts thrashing [7] [16] [27]. In particular, admission control in mixed workload environments aims not only to avoid accepting more work than a database system can effectively process, but also to allow arriving requests to achieve their desired performance objectives.

We divide workload admission control techniques into two types, namely, *threshold-based admission control* and *prediction-based admission control*, as shown in Figure 1. Threshold-based techniques specify the upper limit of a threshold, such as a system parameter, under which an arriving query can be admitted. This type of technique is widely used in workload management facilities provided in commercial DBMSs [30] [50] [72]. As described in Section 2, thresholds are used for controlling a request's admission, and the typical thresholds used are query cost and the number of concurrently running requests. The query cost thresholds dictate that if a newly arriving query has estimated costs greater than the threshold, then the query is rejected, otherwise it is admitted. The MPL threshold dictates if the number of concurrently running requests reaches the threshold, then no new requests are admitted into the system. Workloads may be associated with different sets of threshold values based on a specific admission control policy. Requests with higher priorities can be admitted into the system for execution. The admission control policy may also specify different thresholds for various operating periods, for example during the day or at night.

As an alternative to using system parameters, researchers have proposed threshold-based techniques that rely on performance or monitor metrics, such as the conflict ratio, the transaction throughput in time intervals of the recent past, and system performance indicators. The conflict ratio [56] is the ratio of the total number of locks

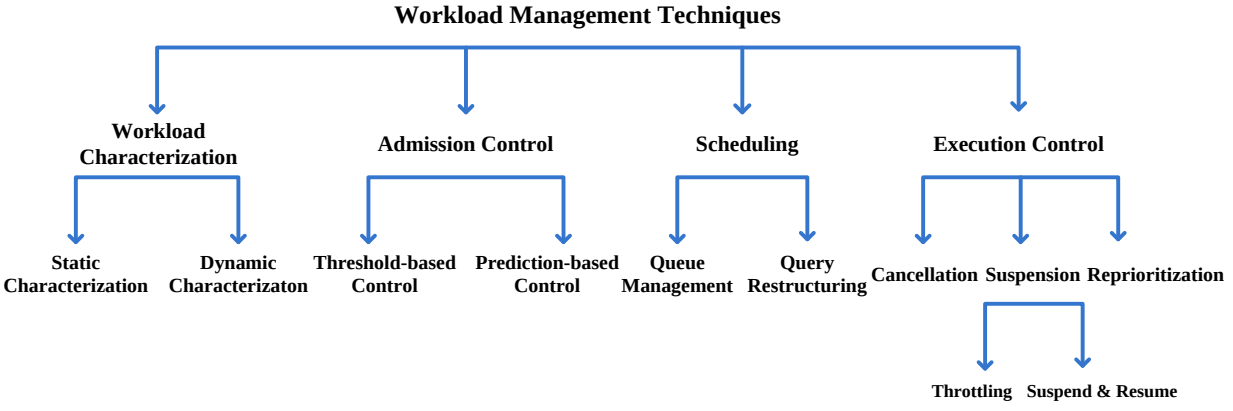

Fig. 1. Taxonomy of Workload Management Techniques for DBMSs

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2767044, IEEE Transactions on Knowledge and Data Engineering

that are held by all transactions in the system and total number of locks held by active transactions. If the conflict ratio exceeds a (critical) threshold, then new transactions are suspended, otherwise they are admitted. The approach that uses the transaction throughput [26] is a feedback method. The approach measures the transaction throughput over time intervals. If the throughput in the last measurement interval has increased (compared to the interval before), more transactions are admitted; if the throughput has decreased, fewer transactions are admitted. The indicator approach [79] [80] uses a set of monitor metrics of a DBMS to detect the performance failure. If the indicator's values exceed pre-defined thresholds, low priority requests are no longer admitted. The basic idea of each approach is to monitor some metric or a set of metrics that indicate the current degree of resource contention in the system, and to react to changes based on specified admission control policies. A summary of the threshold-based approaches used for workload admission control is shown in Table 2.

TABLE 2
Summary of the Approaches Used for Workload Admission Control

Threshold	Type	Description
Query Cost [9] [50] [72]	System Parameter	If an arriving query's estimated cost is greater than the threshold, the query's admission is denied, otherwise, accepted.
MPLs [9] [50] [72]	System Parameter	If the number of concurrently running requests in a database system has reached the threshold, an arriving request's admission is denied, otherwise, accepted.
Conflict Ratio [56]	Performance Metric	If the conflict ratio of transactions in a database system exceeds the threshold, new transactions are suspended, otherwise, admitted.
Transaction Throughput [26]	Performance Metric	If the system throughput in the last measurement interval has increased, more transactions are admitted, otherwise fewer transactions are admitted.
Indicators [79] [80]	Monitor Metrics	If the actual values exceed the pre-defined thresholds, low priority requests are delayed, otherwise they are admitted.

Prediction-based techniques attempt to predict the performance behavior characteristics of a query before the query begins running [21] [23] [42]. These techniques build prediction models for queries using machine-learning approaches. Ganapathi *et al.* [21] find correlations among the query properties, which are available before a query's execution, *e.g.*, the query's SQL statement, the query's execution plan produced by the query optimizer, and query performance metrics, *e.g.*, elapsed time and disk I/O. They use the statistical relationships to predict the performance of newly arriving queries that have the same properties. Gupta *et al.* [23] build a decision tree based on a training set of queries, and use the decision tree to predict ranges of the new query's execution time. Apart from being applied in workload management, this type of technique can be applied in other areas, such as system capacity planning [76].

3.3 Scheduling

As introduced in Section 2, scheduling techniques for workload management involve sending requests to the database execution engine in a particular order that ensures the requests meet their desired performance objectives and also maintains the database system running in a normal (or optimal) state. In today's commercial DBMSs, workload management facilities do not support request scheduling although thresholds, such as MPLs, may be used for implementing certain functionalities of request scheduling. As described previously, MPLs can be employed to manage the load in a database system and therefore maintain the system in a normal state. However, the threshold-based control is a static mechanism. In a dynamic environment, the threshold-based scheduling can result in the database server running in an under-loaded or over-loaded state and cause the requests to miss their required performance objectives.

In contrast with manually setting thresholds to control the load on a database server, current studies have proposed a set of scheduling approaches. In the scheduling class of our taxonomy, the typical techniques presented in the recent research literature can be divided into two types, namely, *queue management* and *query restructuring*, as shown in Figure 1.

The main features of queue management techniques are the determination of execution order of requests based on the properties, such as resource demands, priorities, and performance objectives, as well as a scheduling policy [2] [18] [24] [60] [69]. After passing through an admission control (if any), requests are placed in a wait queue or classified into multiple wait queues according to their performance objectives and/or business priorities (a batch workload may also be treated as a queue, in which a set of requests wait). A scheduler then orders requests from the wait queue(s). The typical approaches include using a linear programming based scheduling algorithm [2] to determine an execution order for all requests in a batch workload, or evaluating the admitted requests queued in the wait queue(s) using a utility function [60] or a rank function [24]. The algorithms and functions take the request's estimated costs and performance objectives as well as business priorities as input and, based on the returned values, the scheduler determines the execution order for the requests. Before the requests can be released from the wait queue(s) to the database execution engine to execute, the total available system resources need to be estimated in order to keep the system running in a normal state. Specifically, the total costs of executing requests should not exceed the database system's currently acceptable cost limits [60]. In order to maintain this constraint, studies show that queuing network models [35] [40] or a feedback controller [17] [28] in conjunction with analytical models may be applied [59] [69] [70]. The models and the controllers attempt to dynamically predict the MPLs for each type of being released requests and keep the system running in a normal state.

Query restructuring techniques [36] decompose a query into a set of small queries, thus the individual queries each being smaller and less complex than the whole.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2767044, IEEE Transactions on Knowledge and Data Engineering

In the context of query scheduling in workload management for DBMSs, a set of decomposed queries can then be put in a queue and scheduled individually. In releasing these queries for execution, no short queries will be stuck behind large queries and no large queries will be required to wait in the queue for long periods of time. By restructuring the original query, the work is executed, but with a lesser impact on the performance of the other requests running concurrently. The approaches [6] [54] to query restructuring involves decomposing a large query execution plan into a series of sub-plans that follow a certain execution order to produce results equivalent to the original query.

3.4 Execution Control

Execution control aims to lessen the impact of executing work on other requests that are running concurrently. We divide the request execution control techniques into three types, namely, *query reprioritization*, *query cancellation* and *request suspension*. Request suspension techniques are further divided into *request throttling* and *query suspend-and-resume*, as shown in Figure 1. The main features of these types of techniques are discussed in this section.

Query reprioritization involves dynamically adjusting the priority of a query as it runs, thereby resulting in resource reallocation [5] [9] [14] to the query according to its new priority. A query's priority adjustment can be dynamically made through adjusting the priority of its workload (at the business level) during execution. Normally, the priority of an end user's query determines the resource access priority of the query. That is, high priority queries have greater access to shared system resources, while low priority queries being given lower priority access to the resources. Priority aging is a typical reprioritization mechanism implemented in commercial DBMSs. The approach dynamically changes the priority of shared system resource access for a request as it runs. When the running request tries to access more rows than its estimated row counts or executes longer than a certain allowed time period, the request's service level will be dynamically degraded, such as from a high level to a medium level, thus reducing the amount of resources that the request can access. Thresholds, such as execution time or the number of returned rows, are incorporated in the approach to adjust the priority of a running request. The events of the threshold violation trigger the adjustment of a request's priority level, and, in turn, adjust the priority of shared system resource access for the request when the request's actual performance behavior violates the thresholds.

From the research literature, an approach proposed for query reprioritization is resource allocation driven by a specified workload importance policy [4] [46] [78]. In this approach, certain amounts of shared system resources are dynamically allocated to competing workloads according to the workload's business importance levels. High business importance workloads are assigned more resources, while low business importance workloads being assigned fewer resources. The amount of shared system resources assigned to a workload can be dynamically changed to respond to changes in the workload importance level as it runs. To enforce the workload business importance policy to resource allocation among competing workloads, utility functions [34] [75] are used to guide the dynamic resource allocation processes, and economic concepts and models [15] [20] are employed to potentially reduce the complexity of the resource allocation problem [58] [78]. The approach shows that more shared system resources can be dynamically allocated to higher business important workloads than the ones with lower business importance during run time.

Query cancellation is widely used in workload management facilities of commercial databases [30] [61] [72] to kill the process of a running query. When a running query is terminated, the shared system resources used by the query are immediately released, and the impact of the query on the performance of concurrently running requests is directly eliminated. The terminated query may be re-submitted to the system for later execution based on a query execution control policy [37] [80]. Like thresholds incorporated in the query reprioritization techniques to trigger the adjustment of a request's resource access priority, a query cancellation procedure can be automatically invoked by the system when a query's running time or consumed shared system resources exceeds threshold values.

Request suspension means slowing down a request's execution. In the *query suspend-and-resume* subclass, the main features of the techniques [10] [12] include terminating a query when it is running, storing the necessary intermediate results and restarting the query's execution at a later time. When a query is suspended, the resources used by the query are released and the impact of the query on the performance of the concurrently running requests is eliminated. The suspended query can be resumed when the database system is less busy. The main feature of request *throttling* techniques [64] [65] [66] is the dynamic manipulation of a request's process as it runs. Instead of terminating a running query and storing its intermediate results, request throttling pauses a running request to slow down its execution, and, therefore, free up some resources used by the request, such as CPU shares and disk I/O. The difference between the throttling and suspend-and-resume types of request suspension techniques is that request throttling pauses the running queries for a certain time, and query suspend-and-resume terminates running queries and continues their execution at a later time. A summary of the approaches used for workload execution control is shown in Table 3.

Apart from using a query execution time threshold to trigger the actions of dynamically controlling a query's execution, a *query progress indicator* can assist the request execution control techniques. It decides whether or not a running query should be controlled based on a specific execution policy. A query progress indicator attempts to estimate how much work a running query has completed and how much work the query will require to finish. This problem has been studied, and a set of techniques have

been proposed in the research literature [11] [41] [43] [45] [55]. Progress indicators keep track of a running query and continuously estimate the query's remaining execution time. The difference between the use of query execution time thresholds and query progress indicators is that thresholds have to be manually set, whereas query progress indicators do not need human intervention and therefore can potentially automate the request execution control.

TABLE 3
Summary of the Approaches Used for Workload Execution Control

Approach	Type	Description
Priority Aging [9]	Reprioritization	Dynamically changes the priority of system resource access for a request as it runs.
Policy Driven Resource Allocation [4] [78]	Reprioritization	Amounts of shared system resources are dynamically allocated to concurrent workloads according to the levels of the workload's business importance.
Query Kill [30] [50] [61] [72]	Cancellation	Kills the process of a request as it runs.
Query Stop-and- Restart [10] [12]	Suspend & Resume	Terminates a query when it is running, stores the necessary intermediate results and restarts the query's execution at a later time.
Request Throttling [64] [65] [66]	Throttling	Pauses the process of a request as it runs.

4 Applications of the Taxonomy

4.1 Commercial Systems

In commercial databases, a workload management facility (or system) is a set of tools or utilities with one or more distinct workload management techniques. The workload management systems include IBM DB2 Workload Manager, Microsoft SQL Server Resource/Query Governor, and Teradata Active Management System. These workload management systems provide good documentation available online for guiding users to use the systems. By distilling the information and applying the taxonomy of workload management techniques shown in Section 3, we discuss these systems and identify the workload management techniques employed in the systems. In this study, we do not discuss all of the workload management facilities implemented in commercial databases, but the typical techniques are covered in the three systems. To describe these workload management systems, an overview of the systems is presented in the following subsections.

4.1.1 IBM DB2 Workload Manager

Since the version 9.5 release, IBM DB2 has integrated a workload management facility, *Workload Manager*, in the DB2 databases for Linux, UNIX and Windows [30]. In the workload management, there are three defined stages, namely *identification*, *management* and *monitoring*, for users to manage complex workloads on a DB2 database server. The *identification* identifies the requests entering the database server, the *management* manages the requests running on the server, and the *monitoring* monitors whether or not performance objectives of the requests have been met, and the database server is being efficiently used. Before applying the stages to make a workload management plan and implemente it, users are suggested to understand request goals. That is, to understand if any SLA exists for the requests, what business priorities of the requests are relative to all of the other work on the system, and what the performance objectives (if any) of the requests are.

A) Identification

In *identification*, *workloads* (database objects) are used in DB2 workload management to identify incoming work. This is implemented based on the source of the work. The source of the work is determined using attributes of database connections, which are assigned when a database connection is established. The attributes include *application name*, *system authorization ID*, *session ID*, and *client user ID* that can be used to uniquely identify a connection. Thus, work coming through a connection can be mapped to a pre-defined *workload*, so in the use of the source of the work, users identify incoming work through its origination. In addition to using connection attributes, incoming work is also identified using *work classes* (database objects), which is created based on the type of incoming work. A *work class* is defined in a *work class set* (database objects). A work class has the incoming work with the same type, such as RAED, WRITE, or, DML, DDL, LOAD, CALL and ALL (all types). Work classes can also apply predictive identification to the DML type of work. The predictive elements include *estimated costs* and *estimated return rows*. Users can create a work class, for instance, for all large queries with an estimated cost over 1,000,000 *timerons*, or create a work class for all large queries with estimated return rows more than 500,000.

B) Management

A *service class* is another object in DB2 databases. Service classes are used for defining execution environments where the arriving work runs. Execution environments allocate shared system resources to the work, and create various execution thresholds that detemine how the work is allowed to execute. All work must run in a service class, and the *workload definition* is used for assigning incoming work to a service class. When a service class is created, its resource access priorities are also created, which include *agent priority*, *prefetch priority*, *buffer pool priority* and *external WLM tag*.

The agent priority is set for a CPU priority level of all agents that work in a service class, and is relative to the agent priority of all other DB2 agents. The prefetcher priority is set for prefetch requests that are generated in a service class. Agents send read-ahead requests to the database prefetch queue, and then the prefetchers take these read-ahead requests from the queue. High-priority prefetch requests are processed before medium-priority prefetch requests, which, in turn, are processed before low-priority prefetch requests. Setting the buffer pool priority of a service class can influence the proportion of pages in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2767044, IEEE Transactions on Knowledge and Data Engineering

the buffer pool that are used by requests in the service class. Increasing the buffer pool priority potentially increases the proportion of pages in use by the requests in a particular service class. The external WLM tag allows a workload to have some of its resources controlled by the AIX Workload Manager [8]. A service class can be divided into multiple service subclasses. The service class remains the highest tier for work, and service subclasses are the place where arriving work runs. A subclass can only be defined under a service class, that is, a subclass cannot be defined under another subclass.

At times, a request's execution behaviour exceeds expectations. As an example, a query surprisingly returns hundreds of thousands of rows and consumes a large amount of (I/O) resources at the expense of all of the other work running in the system. In DB2 workload management, *thresholds* (database objects) are used to look for this type of exception and to trigger actions when the thresholds are violated. These thresholds include *Elapsed Time*, *Estimated Cost*, *Rows Returned*, *Concurrent Workload Activities* and *Concurrent Database Activities*. Actions that can be taken when a threshold is violated depend on the threshold's definition. The actions include *collect data*, *stop execution*, *continue execution*, and *queue activities*.

In a service class (with its created thresholds), the resource access priorities of a query can be dynamically changed by moving the query from one subclass to another service subclass (under the same service class). More resources can be accessed if the priorities of the new service subclass are higher, and fewer resources are available if the priorities of the new service subclass are lower. A query is moved between two service subclasses when the thresholds are violated based upon the pre-defined maximum usage of a specific resource, such as CPU time or return rows. The priority change (priority aging) is triggered when the threshold violation is detected. After a query is mapped to a new service subclass, it continues to run with the new resource constraints applied.

C) Monitoring

The *monitoring* provides users the access to real-time operational data, such as a list of running workload occurrences, the queries running within a service class, and the averaged response time. The method for accessing the real-time monitor data is through using *table functions*. Table functions provide users capabilities to create applications (or write queries) to query data as if it were a table in the databases. Statistical information is available at a few levels, which include *service classes*, *service subclasses*, *workloads*, *work action sets* and *queues*. Besides the table functions, event monitors, such as *activity event monitor*, *threshold violations event monitor* and *statistics event monitor*, are used to capture monitor information. The *activity event monitor* captures information about individual queries in a service class, workload, or work class. The *threshold violations event monitor* captures information as a threshold is violated. It indicates what threshold was violated, what query was the source of the exception, and what action was taken when it occurred. The *statistics event monitor* captures detailed query information by collecting aggregated data, such as the number of queries completed and the averaged execution time.

4.1.2 Microsoft SQL Server Resource/Query Governor

Microsoft SQL Serve provides users a workload management facility, *Resource and Query Governor*, for managing workloads and system resources. *Resource Governor* [50] enables users to manage workloads and resources through specifying limits on resource usage of arriving requests. *Query Governor Cost Limit Option* [51] is used for specifying an upper limit on execution time, under which a query can run. In a SQL Server environment, if Query Governor Cost Limit is specified with a nonzero and nonnegative value, the *query governor* will disallow execution of any arriving query that has an estimated execution time exceeding the value, while, specifying zero (the default value) meaning all queries can run without any time limitation. Query Governor Cost Limit can be applied to the server wide or to per database connection. Resource Governor manages complex workloads present on SQL Server by differentiating the workload requests and allocating shared system resources to the requests based on the limits that users specify. Resource Governor consists of three main components, namely *resource pool*, *workload group* and *classification*.

A) Resource Pools

A *resource pool* represents physical resources (CPU and Memory) of the server. A resource pool has two portions. One partion does not overlap with other pools, which enables a minimum resource reservation in the resource pool. The other partion is shared with other pools, which supports maximum resource consumption on the server. In the use of the resource pool, the resources are allocated by specifying MIN and MAX. MIN represents the minimum guaranteed resource available in the resource pool, and MAX represents the maximum resources of the pool. MIN and MAX are set for the two resources (CPU and Memory), respectively. The sum of MIN across all resource pools cannot exceed 100 percent of the server resource. MAX can be set anywhere in the range between MIN and 100 percent inclusive. The shared portion of a pool indicates that a certain amount of resources can go if the resources are available. However, when the resources are used, *e.g.*, they go to a specified resource pool, they become not shared any more. These shared resources improve resource utilization in the cases, where there are no requests in a pool, and the resources configured to the pool can be freed up for other pools.

In the use of Resource Governor, two resource pools, *i.e.*, *internal* and *default*, are predefined. The *internal* pool represents the resources used by SQL Server itself. This pool contains only the *internal workload group*, and the pool is not alterable. Resource consumption by the *internal* pool is not restricted. That is, the workload in the pool are considered critical for server functions, and the *internal* pool is allowed to "pressure" other pools even if it means the violation of limits set for those pools. The *de-*

fault pool contains *default workload group*. The *default* pool cannot be created or dropped but it can be altered. The *default* pool also contains user defined workload groups in addition to the *default workload group*.

B) Workload Groups

The *workload group* serves as a container for session requests that are similar according to the classification criteria (the criteria are applied to each request as it arrives). The workload group provides users the capabilities of monitoring resource consumption and applying a uniform policy to all the requests in the group. In the use of Resource Governor, two workload groups, *i.e.*, *internal* and *default*, are predefined. Users cannot change anything classified as the *internal workload group*, but users can monitor the workload group. Incoming requests are classified into the *default workload group* if there are no criteria defined to classify the incoming requests, or if there is an attempt to classify the requests into a nonexistent group, or there is a failure with the classification. Resource Governor also supports the creation of user defined workload groups. A user-defined workload group is assicated with a resource pool, and it can be moved from one resource pool to another.

C) Classification

Classification is used by Resource Governor to differentiate the incoming session requests, which is based on a set of user-written criteria defined in a classification function. The results of the function logic enable Resource Governor to classify session requests into an existing workload group. Before a classification function can be used, users need to create and register the classification function and update the Resource Governor configuration. After the configuration change being applied, the Resource Governor classifier can use the workload group name returned by the function to send a new request to the appropriate workload group. The characteristics and behaviors of a classification function is that the function is defined within the server scope, it is evaluated for each new session, the function is designated as a classifier, and it gives a workload group the context of a session.

D) Monitoring

Resource Governor provides users monitoring capabilities to obtain execution statistics for *workload groups* and *resource pools*. Performance counters, *e.g.*, *Workload Group Stats* and *Resource Pool Stats*, are used to collect workload group and resource pool statistics. The first counter reports statistics for each active workload group, such as the number of active requests, and the number of blocked requests. The second counter reports statistics for each active resource pool, such as the number of memory grants that occurs in the resource pool per second, and the amount of memory that is used by the resource pool. In addition to using the performance counters, Resource Governor introduces events, *e.g.*, *CPU Threshold Exceeded*, *Pre Connect Starting* and *Pre Connect Completed* to indicate when Resource Governor detects a query, which has exceeded the CPU threshold value, when Resource Governor Classifier starts execution, and when the classifier finishes execution. Resource Governor also introduces dynamic management views, *e.g.*, *Resource Governor Workload Groups*, *Resource Governor Resource Pools* and *Resource Governor Configuration*, to return current statistics and configuration for workload groups and resource pools.

4.1.3 Teradata Active System Management

Teradata Active System Management (ASM) is a set of tools and utilities used for managing complex workloads on Teradata databases [71] [72]. Teradata ASM consists of four main components, namely *Teradata workload analyzer*, *Teradata dynamic workload manager*, *Teradata manager* and *Teradata regulator*. The first three are the graphical user interface (GUI) tools of Teradata client applications, and the last one, *Teradata regulator*, is a component inside Teradata databases. The capabilities of Teradata ASM include analyzing and defining workloads, regulating system resources, monitoring performance and identifying abnormalities.

A) Teradata Workload Analyzer

The *Teradata workload analyzer* (WA) is a tool that provides users recommendations on *workload definition* and *operating rules*. A workload definition is used for defining workloads, and the operating rules are used for helping databases meet service level goals (SLGs). By analyzing the data of database query log (DBQL), Teradata WA provides users workload recommendations, which include workload definitions, workload SLGs, and the mapping between workloads and resource allocation groups. By using Teradata WA, users can also establish operating rules based on the current system configurations and the analysis of workload statistics.

A workload analysis process includes collecting query DBQL, specifying dimensions to analyze the collected data, and grouping the queries to form candidate workloads. Teradata WA recommends candidate workload definitions based on the workload analysis. In the use of the DBQL analysis, users can further refine the candidate workload definition by either merging with another candidate workloads or splitting a candidate workload into two or more separate candidate workloads.

B) Teradata Dynamic Workload Manager

The *Teradata dynamic workload manager* (DWM) supports detailed creation and management of workload definitions. The workload definition is a set of rules that describe a class of queries for the purpose of appropriate resource allocation based on performance objectives. The rules apply filters, throttles, and classification on queries to regulate their execution behaviors, and determine exceptions. Teradata DWM provides users a GUI tool to create rules and to manage workloads based on system states or environment events.

Rules define how a Teradata database manages workloads. As described above, three types of rules, namely *filters*, *throttles* and *workload definitions*, are used for managing workloads. The *filters* reject unwanted logon or queries before execution. There are two types of filters, namely *object access filters* and *query resource filters*. The *object access filters* limit access to specific database objects

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2767044, IEEE Transactions on Knowledge and Data Engineering

for certain or all types of SQL requests. The *query resource filters* limits queries that are estimated to access "too many" rows, take "too long" to complete, or perform certain types of joins.

The *throttles* (*i.e.*, concurrency rules) limit the number of active sessions, queries or utilities on a Teradata database. There are two kinds of throttles, namely, *object throttles* and *utility throttles*. The *object throttles* limit the number of queries executed simultaneously against a database object. The *utility throttles* enforce concurrency limits on the database utilities, such as load, export and restore, that run simultaneously. The filter and throttle rules can be applied to the database wide or to per workload.

The *workload definitions* specify how a Teradata database regulates queries, which includes *classification criteria*, *execution behaviors*, *exception criteria* and *actions*, and *SLGs*. The *classification criteria* determine whether or not a query can be assigned to a certain workload. The criteria include "who", which specifies the source of the request, such as user id, account, application, and client IP address, "where", which specifies objects being accessed, such as tables, views, and databases, and "what", which specifies characteristics of the request, such as estimated processing time and join types. The *execution behaviors* specify the mapping of a workload to a priority level and a resource allocation group. The *execution behavior* also defines a workload concurrency throttle, which specifies how many queries can be executed at one time under the workload definition. When the threshold is exceeded, new queries are placed on a delay queue. The *exception criteria* include a set of conditions, such as high IO skew or too much CPU processing time, which are determined after a query begins execution. The *exception actions* specify what actions to take when an exception occurs. *SLGs* specify performance objectives of workloads.

C) Teradata Manager

The *Teradata manager* is a GUI tool that helps users monitor a Teradata database and visualize real-time performance and historical trends. The *Teradata manager* includes *dashboard workload monitor* and *workload trend analysis*. The dashboard workload monitor provides a view of current and recent historical workload status, as well as the option of the change of the workload definition that is assigned to the current session or all sessions. The information provided by the *dashboard workload monitor* includes CPU usage per workload, number of active sessions per workload, request arrival rate of a workload (in the last collection period), the number of complete requests per workload, response time of requests in a workload, the number of requests violate SLGs in a workload, the number of requests currently on delay queue per workload, list of session numbers, and workload names. The *workload trend analysis* lists workload definitions according to various user defined criteria, and reports workload reources usage trends.

D) Teradata Regulator

The *Teradata regulator* is a proactive tool for managing the system performance. Requests submitted to the system are classified into an appropriate workload and managed based on the workload's operating rules, such as throttles, resource priorities, and exception management. The regulator also monitors queries as they run to check for exception conditions, such as CPU time, IO count, CPU to disk ratio, CPU or IO skew, response time and blocked time, and then handles them according the rules defined with the workload definition configurations.

4.1.4 Evaluation of the Workload Management Systems

A summary of the workload management systems described in the previous subsections is shown in Table 4. The classification of the techniques employed in the systems is processed using the taxonomy of workload management techniques depicted in Section 3. The workload management systems provide rich sets of monitoring tools. Typically, *monitoring* is a separate component in a DBMS, so the taxonomy of workload management techniques does not examine and classify the monitoring tools. As none of the systems implements any *scheduling* technique, the *scheduling* class is not discussed and presented in the table.

TABLE 4
SUMMARY OF THE WORKLOAD MANAGEMENT SYSTEMS

Workload Management Systems	Workload Characterization	Admission Control	Execution Control
IBM DB2 Workload Manager [30]	Based on the source or type of incoming work, workloads are created	Thresholds are used to manage request concurrent levels at the workload or the database level	Service classes are used to allocate requests resources and thresholds are used to monitor and control the request's execution behaviour
Microsoft SQL Server Resource/Query Covernor [50] [51]	Using classification functions, incoming work is differentiated in workload groups	Query Governor is used to evaluate arriving queries based on their cost limits	Resource pools dynamically allocate resources and performance counters, thresholds and views are used to monitor requests execution behaviour
Teradata Active System Management [71] [72]	Teradata workload analyzer recommends a workload for a class of queries	Filters & throttles are used to reject unwanted requests and to control request concurrent levels	Teradata DWM allocates resources to requests based on the workload definition, and rules are used to monitor and control the request's execution behaviour

IBM DB2 Workload Manager provides the *identification*, *management* and *monitoring* stage to manage complex workloads on DB2 databases. In the identification stage, the *workload* and *work class* (set) are used to identify incoming work based on the source and the type of the incoming work. In the management stage, the *service class* is used for providing an execution environment to defined workloads. The execution environment allocates shared system resources and creates thresholds to manage the workload execution. Thresholds are used for detecting exceptions and triggering actions if the thresholds are violated. Actions include rejecting a request admission, stopping a request execution and conducting priority ag-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2767044, IEEE Transactions on Knowledge and Data Engineering

ing, *e.g.*, downgrading resource access priorities of a running workload. In monitoring stage, the performance of running workloads is monitored. By applying the taxonomy of workload management techniques, we examine IBM DB2 Workload Manager and identify the employed techniques in the workload management system, which include static workload characterization, threshold-based (costs, types and MPLs) admission control, and execution control with query re-prioritization (dynamica resource re-allocation) and query cancellation.

Microsoft SQL Server provides Resource Governor to manage complex workloads on a Microsoft SQL Server. The resource governor consists of *resource pools*, *workload groups* and *classification* components. A resource pool has two portions. One portion does not overlap with other pools, which reserves minimum amount of resources. The other portion is shared with other pools, which supports maximum possible resource consumption. A workload group serves as a container for requests that are similar according to the classification criteria. Classification is used to differentiate the incoming requests, which is based on a set of user-written classification functions. The Query Governor Cost Limit option is used for specifying the upper limits of query execution time. The query governor disallows execution of incoming queries that have an estimated execution time exceeding the cost limit. By applying the workload management technique taxonomy, we examine Microsoft SQL Server Resource Governor and identify the employed techniques in the system, which include static workload characterization and execution control with dynamic resource reallocation. The Query Governor Cost Limit option employs threshold-based admission control technique.

Teradata Active System Management is used for managing complex workloads on a Teradata database. Teradata ASM consists of *Teradata workload analyzer*, Teradata *dynamic workload manager*, *Teradata manager* and *Teradata regulator* components. The main component of Teradata ASM is *Teradata dynamic workload manager*. There are three categories of rules, namely *filters*, *throttles* and *workload definitions*, are used in *Teradata dynamic workload manager* to provide workload definition and workload controls. The filters reject unwanted logon and query requests before they are executed. The throttles limit the number of active sessions, query requests, or utilities on a Teradata database. The workload definition specifies a workload's classification criteria, execution behaviors, exception criteria and actions and service level goals. To examine Teradata Active System Management, we apply the workload management technique taxonomy to identify the employed techniques in the workload management system. The techniques include static workload characterization, threshold-based admission control, and execution control with query cancellation.

4.2 Techniques in Research Literature

In contrast with the use of thresholds as a main execution control mechanism in commercial databases, dynamic workload scheduling and execution control approaches have been developed in research. In the following subsections, we describe typical techniques proposed in the research literature and classify them into a technique class (and a subclass) by applying the taxonomy of workload management techniques. A description of the techniques is presented below.

4.2.1 Query Scheduling Techniques

Niu *et al.* [60] propose a query scheduler to manage the execution order of multiple classes of queries in order to achieve the workload's service level objectives (SLOs). The query scheduler is built on an IBM DB2 database system and uses DB2 Query Patroller [30], the DB2 query management system, to intercept arriving queries, acquire information of the queries, determine a suitable order of execution, and then release the queries to the database engine for execution. The Query Scheduler has two main processes, namely the workload detection process and the workload control process.

The workload detection process classifies arriving queries based on their SLOs, which include the query's performance goals and business importance, and monitors performance to detect whether or not queries are meeting their performance goals. If the queries miss their performance goals, the query scheduler calls the workload control process to change the mix of queries in order to allow the more important queries to meet their performance goals. The query scheduler's workload control process implements a cost-based approach and periodically generates new plans to respond to the changes in the mix of arriving requests. A scheduling plan is generated based on the cost limits of the service classes to which arriving queries belong, the arriving queries' performance goals and the database system's available capacity. The cost limit of a service class is the allowable total cost of all concurrently running queries belonging to the service class. The query scheduler uses utility functions to estimate how effective a particular cost limit will be in achieving performance goals. An objective function, which is defined based on the utility functions, is used to measure if a scheduling plan is achieved, and an analytical model is used to predict the system performance when a scheduling plan is applied.

4.2.2 Request Throttling Techniques

A) Utility Throttling

Parekh *et al.* [64] propose a database utility throttling approach to limit the impact of on-line database utilities on user's work. Database utilities may include statistics update, index rebuild, database backup and restore, and data reorganization. These are essential operations for a database system's good performance as well as operation and maintenance, but when they are executed on-line, the database utilities can significantly degrade the performance of production applications. The authors attempt to show how these on-line utilities can be controlled so the performance degradation of production applications can be managed at an acceptable level.

In their approach, a self-imposed sleep is used to slow down (throttle) the on-line utilities by a configurable amount. All work present on a database system is divided

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2767044, IEEE Transactions on Knowledge and Data Engineering

into two classes, namely utilities and production applications. The system monitors the performance of the production applications and reacts according to high-level policies to decide when to throttle the utilities and estimates the appropriate amount of throttling. The performance degradation of production applications is determined by comparing the current performance with the baseline performance acquired by the production applications. The authors assume a linear relationship between the amount of throttling and system performance and use a Proportional-Integral controller to control the amount of throttling, and a workload control function translates the throttling level into a sleep fraction for the on-line utilities.

B) Query Throttling

Using the work of Parekh *et al.*, Powley *et al.* [65] [66] propose an autonomic query throttling approach to dynamically throttle large queries in a database system based on high-level business objectives. Query throttling is a workload execution control approach that slows the execution of large queries to free up resources for other work running concurrently in the system to achieve the required performance goals. In their approach, a self-imposed sleep is used to slow down (throttle) long-running queries, and an autonomic controller automatically determines the amount of throttling for large queries needed to allow other concurrently running queries to meet their performance goals.

Powley *et al.* use autonomic computing principles [22] to build two different types of controllers, a simple controller and a black-box model controller, and compare their effectiveness in determining that high priority workloads meet their goals. The simple controller is based on a diminishing step function, and the black-box model controller uses a system feedback control approach. The authors also develop two query throttling methods, which are called constant throttle and interrupt throttle. The constant throttling method involves many short pauses, which are consistent and evenly distributed throughout a query's run time, thus slowing the query's execution. The pause length is a parameter that can be defined by a user, but the number of pauses is determined by the amount of throttling. The interrupt throttling method involves only one pause throughout a query's run time, and the pause length is determined by the amount of throttling.

4.2.3 Query Suspension and Resumption Techniques

Chandramouli *et al.* [10] propose a query suspend-and-resume approach to controlling long-running and resource-intensive analytical queries. It is a query execution control technique that attempts to provide DBMSs with the ability to quickly suspend long-running and low-priority queries when high-priority queries arrive, and resume the suspended queries when the high-priority work has completed. To achieve this goal, the lifecycle of traditional query execution is augmented with two new phases, called suspend and resume, that are triggered on demand.

Once the database query optimizer chooses an execution plan for a query, the query enters its execution phase. Upon receiving a suspension request, the query enters its suspension phase. A SuspendedQuery data structure is produced in this phase, which encapsulates all the information needed to resume the query later. This structure may be written on disk. A suspend cost is incurred during the query's suspension, but it needs to be low. After suspension, all of the query's resources are released. When the DBMS is ready to resume the query, it enters the resume phase. The SuspendedQuery structure is read back into memory and the query's execution state is set to the suspend point, so the execution phase can continue from where it was interrupted. In their approach, asynchronous checkpointing is proposed in the execution phase, with each operator checkpointing independently of others in the query plan. In the suspend phase, a new suspend strategy, GoBack, is proposed as an alternative to DumpState, so an operator writes only its current control state to the SuspendedQuery structure at the time of suspend. Although GoBack incurs a lower suspend cost than DumpState, it can result in a higher resume cost than DumpState in the resume phase. In their approach, authors use mixed-integer programming to find the optimal suspend plan that minimizes the total overhead of suspend/resume while meeting a given suspend cost constraint.

4.2.4 Query Kill and Resource Reallocation Techniques

Krompass *et al.* [39] propose an automated workload execution control approach for Business Intelligence workloads on a data warehouse. Managing BI workloads can be a challenge as BI queries exhibit large variances in response times, resource demands and may have different SLOs. The workload execution control approach consists of two main components, a query execution controller and a set of query execution control actions. The execution control component is implemented with a rule-based fuzzy logic controller, and the query execution control actions include query reprioritize, kill and resubmit after kill. The workload execution control approach is used to manage problematic queries that consume a large amount of resources and run for a long time.

The authors use a fuzzy logic controller as they believe the fuzzy logic paradigm can address issues including classifying queries based on the expected behavior where the queries' execution times are not entirely predictable; governing queries' execution where there are numerous factors that have to be considered; complete knowledge about the state of a data warehouse and the queries running in the system is not available due to the complexity of the system. In their workload execution control approach, the controller uses information gathered at runtime to manage the queries concurrently running in a database system. The monitored metrics include priority, number of query cancellations, operator progress, resource contention *etc.*, in which the priority of a query has an impact on the resource allocation. Based on these metrics, the controller can impose several control actions on

the problematic queries in order to control workloads. With the reprioritize action a query is re-prioritized and its resources are redistributed immediately among the other queries according to the priorities of the individual queries. The kill action kills a running query and immediately frees the resources used by the query. Any intermediate results generated during the execution of the query are disposed. The kill-and-resubmit action kills a running query and the query is queued again for subsequent execution.

4.2.5 Evaluation of the Workload Management Techniques

In this section, we apply the taxonomy to classify the research techniques discussed in the previous subsections.

TABLE 5
SUMMARY OF THE WORKLOAD MANAGEMENT TECHNIQUES

Proposed Techniques	Technique Classes	Features	Objectives
Niu *et al.* [60]	Admission Control & Scheuling	Intercepting arriving queries, acquiring their information, and determining an execution order	Achieving a set of service level objectives for multiple concurrent workloads
Parekh *et al.* [64]	Execution Control, throttling	A self-imposed sleep slows down online utilities; a Proportional Integral controller determines the amount of throttling	Maintaining performance of running workloads at an acceptable level
Powley *et al.* [65] [66]	Execution Control, throttling	A self-imposed sleep slows down large queries; a step function and a black-box model determine the amount of throttling	Meeting the service level objectives of high-priority requests
Chandramouli *et al.* [10]	Execution Control, suspend and resume	Query execution is augmented with suspend and resume phases that are triggered on demand	Achieving high performance for high-priority requests
Krompass *et al.* [39]	Execution Control, query cancellation and reprioritization	Cancelling or reprioritizing low-priority and long-running queries	Achieving high performance for high-priority requests

The query scheduler technique proposed by Niu *et al.* can be classified into query scheduling and admission control classes, as the technique attempts to determine the execution order for arriving queries, and to maintain the optimal number of concurrent requests in a database system based on the performance goals, cost limits and priorities of the arriving queries. The database utility throttling approach proposed by Parekh *et al.* and the large query throttling approach proposed by Powley *et al.*, respectively, can be classified into query throttling technique subclass, as the techniques attempt to slow down the execution of a running request. Although a database utility is not a query, the actual meaning of "query throttling" here is the database request throttling. The query suspension and resumption approach proposed by Chandramouli *et al.* is a typical technique in query suspend & resume subclass, as it attempts to stop the process of a running query and restart it at a later time. The workload execution control approach proposed by Krompass *et al.* can be classified into query kill and resource reallocation subclasses, as the typical query kill and resource reallocation techniques shown in Section 3 are used in the workload execution control approach. A summary of the research techniques described above is shown in Table 5.

5 CONCLUSIONS AND FUTURE RESEARCH

5.1 Summary

In this paper, we present a systemic study of workload management in DBMSs. We surveyed workload management systems implemented in today's commercial DBMSs and techniques proposed in the recent research literature. We propose a taxonomy of workload management techniques to classify workload management techniques and identity the techniques employed in a workload management system. The taxonomy categorizes workload management techniques into four major technique classes, namely workload characterization, query admission control, query scheduling and query execution control technique classes. In a workload management technique class, the techniques may be further divided into subclasses based on their distinct technique mechanisms. We also introduce the underlying principles of workload management technology used by today's commercial DBMSs, which are outlined as defining performance objectives for arriving queries based on a given SLA, identifying the arriving queries present on a data server, and imposing controls on the queries to manage their behaviors in order to achieve the performance goals.

In the taxonomy of workload management techniques, we show that the typical technique used in the workload characterization technique class is workload definition associated with resource allocation. The typical technique used in the query admission control techniques class is setting thresholds for queries. The typical techniques used in the query scheduling technique class are managing query waiting queues and query restructure, and in the query execution control technique class, the typical techniques are query suspension, resource reallocation and query kill, in which the query suspension type techniques can be further divided into query throttling and query suspend-and-resume subtypes.

5.2 Open Problems

Despite the efforts of researchers and developers in both academia and industry to provide facilities to effectively manage highly varied and frequently changing workloads, workload management in DBMSs is still an open research area. There are several issues that need to be explored and addressed for today's workload management systems, which include automatically choosing and applying appropriate techniques to manage customers' requests during execution, dynamically estimating available system capacity and execution progress of running queries as well as reducing the complexity of a workload management system's operation and maintenance.

Varied workload management techniques, as discussed previously, have been developed and implemented in most major commercial DBMSs, but it is un-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2767044, IEEE Transactions on Knowledge and Data Engineering

clear what techniques are appropriate and should be chosen and applied to be most effective for a particular workload executing on the DBMSs under certain particular circumstances, or how the multiple techniques can be well coordinated to manage performance of all running workloads to meet their required performance goals. For example, consider a data server, where there is an ad hoc workload present while a large number of important transactional requests are arriving. The ad hoc workload's execution may need to be restricted in order to free up sufficient shared system resources for the important requests to reach their performance goals. To restrict the ad hoc workload execution, several approaches may be applied, which include the ad hoc workload's kill, throttle, suspension and resumption, and priority degradation for shared system resource access. It is unclear which techniques are most appropriate and effective under this circumstance. In addition, the interplay among multiple workload management techniques can be difficult to anticipate at runtime.

System capacity estimation is also significant in the workload management process, as all controls imposed on the end user's requests are based on the system state. If the system state of a database server is overloaded, no requests can be admitted and scheduled, while some running requests should have their execution slowed down and release some used resources. The progress estimation of a running query provides the necessary information for the query's execution control. A small query may be queued in a database system for a certain amount time for execution, and the query's spent time in the system exceeds the threshold upper limits of query execution time. If there is less information about the query progress, the query can be treated as a long-running query and killed for releasing shared system resources for more important requests. However the performance of important requests would not be improved as the query was not a big consumer of the resources.

A DBMS is a complex information management system, and it can have hundreds of tuning parameters for performance optimization. With the integration of workload management features, a large number of workload control threshold values must be well understood and set by the system administrators, thus rendering the entire system becomes more complex in terms of operation and maintenance.

5.3 Our Vision, Approach and Future Research

In order to resolve these issues, many researchers and engineers [48] [59] [77] consider that building an automated workload management system for DBMSs is a possible approach. An automated workload management system is a self-managing system, which is capable of automatically controlling complex workloads on a DBMS based on the workload performance goals, actual performance behaviors and the available system resources. To achieve this goal, we envision Autonomic Computing as the most effective approach [22] [29] [32]. The initiative of autonomic computing aims to provide the foundation for systems to manage themselves without direct human intervention in order to reduce the complexity of the computing environment and infrastructure. In this vision, systems manage themselves in accordance with high-level business objectives, and a fully autonomic computing system has the properties of self-configuring, self-optimizing, self-protecting and self-healing. Self-configuring means computing systems are able to automatically configure components to adapt to dynamically changing environments. The functionality of the property allows the addition and removal of system components or resources without system service disruptions. Self-optimizing means that systems automatically monitor and control the resources to ensure optimal functioning with respect to the defined requirements. Self-healing means that systems are able to recognize and diagnose deviations from normal conditions and take action to normalize them. This property enables a computing system to proactively circumvent issues which could cause service disruptions. Self-protecting means computing systems are able to proactively identify and protect from arbitrary attacks.

Although, in the current evolution stage, autonomic computing faces a challenge that no one has yet built a large-scale fully autonomic computing system or prototype, many successful autonomic components have been developed and are proving useful in their own right [33] [49]. In particular, we consider applying autonomic computing principles to build an autonomic workload management system for DBMSs to manage complex workloads based on given high-level business objectives. The autonomic workload management system may include all the typical workload management techniques discussed previously implemented using feedback loop control and utility functions [34] [75]. The feedback loop control consists of four components, which are a monitor that continuously monitors a database system performance, an analyzer that analyzes the database system available capacity and the running query's execution progress, and compares the running query's performance with their required performance goals, a planner that decides what technique is most effective for a running workload under its certain circumstances by applying the utility function, and an effector that imposes the control on the workload. The feedback control loop monitors changes of a database system's performance and running workload's type mix, takes effective actions and keeps the workloads to meet their performance goals [80].

ACKNOWLEDGMENT

The authors wish to thank Dr. Yanto Qiao for editing the first version of this paper.

REFERENCES

[1] A. Aboulnaga and S. Babu. "Workload Management for Big Data Analytics". *Tutorial at the ACM SIGMOD International Conference on Management of Data (SIGMOD '13),* New York, USA. June 2013. pp. 929-932.

[2] M. Ahmad, A. Aboulnaga, S. Babu and K. Munagala. "Interaction-aware Scheduling of Report-generation Workloads". In *Intl. Journal on Very Large Data Bases.* Vol. 20, Issue 4, August 2011. pp. 589-615.

[3] P. Bird and R. Kalesnykas. "Best Practices: Implementing DB2® Workload Management in a Data Warehouse", *a White Paper of IBM.* 2011.

[4] H. Boughton, M. Zhang, W. Powley, P. Martin, P. Bird and R. Horman. "Using Economic Models to Capture Importance Policy for Tuning in Autonomic Database Management Systems". In *International Journal of Autonomic Computing,* Vol. 2, No. 2, 2016.

[5] K. P. Brown, M. J. Carey, and M. Livny. "Managing Memory to Meet Multiclass Workload Response Time Goals". In *Proc. of the 19th Intl. Conf. on Very Large Data Bases (VLDB '93).* Dublin, Ireland. August 24-27, 1993. pp. 328-341.

[6] N. Bruno, V. Narasayya and R. Ramamurthy "Slicing Long-Running Queries". In *Proc. of the VLDB Endowment*. Vol. 3, Issue 1-2, Sept. 2010. pp. 530-541.

[7] M. J. Carey, S. Krishnamurthi and M. Livny. "Load Control for Locking: The Half-and-Half Approach". In *(PODS'90): Proc. of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems.* 1990. pp. 72-84.

[8] S. Castro, N. Tezulas, B. Yu, J. Berg, H. Kim and D. Gfroerer. "AIX 5L Workload Manager (WLM)". *An IBM RedBooks Publication,* June 2001.

[9] W. J. Chen, B. Comeau, T. Ichikawa, S.S. Kumar, M. Miskimen, H.T. Morgan, L. Pay and T. Väättänen. "DB2 Workload Manager for Linux, Unix, and Windows". *An IBM RedBooks Publication,* May 2008.

[10] 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,* Beijing, China, 2007. pp. 557-568.

[11] S. Chaudhuri, R. Kaushik, and R. Ramamurthy. "When Can We Trust Progress Estimators for SQL Queries?" In *Proc. of the 2005 ACM SIGMOD Intl. Conf. on Management of Data.* USA. 2005. pp. 575-586.

[12] S. Chaudhuri, R. Kaushik, A. Pol, and R. Ramamurthy. "Stop-and-Restart Style Execution for Long Running Decision Support Queries". In *Proc. of VLDB '07,* Vienna, Austria, 2007. pp. 735-745.

[13] S. Chaudhuri. "An Overview of Query Optimization in Relational Systems". In *(PODS'98): Proc. of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems.* 1998. pp. 34-43.

[14] B. Dageville and M. Zait, "SQL Memory Management in Oracle9i". In *Proc. of VLDB'02,* Hong Kong, China. 2002. pp. 962-973.

[15] D. L. Davison, G. Graefe. "Dynamic Resource Brokering for Multi-User Query Execution". In *ACM SIGMOD Record,* Volume 24, Issue 2, May 1995. pp. 281-292.

[16] P. J. Denning. "Thrashing: Its Causes and Prevention". In *Proceedings of the AFIPS Joint Computer Conferences.* San Francisco, California, USA. December 1968. pp. 915-922.

[17] Y. Diao, J. L. Hellerstein, S. Parekh, R. Griffith, G. E. Kaiser and D. B. Phung. "A Control Theory Foundation for Self-Managing Computing Systems". *IEEE Journal on Selected Areas in Communications.* Vol. 23, No. 1, January 2005. pp. 2213-2222.

[18] J. Duggan, U. Cetintemel, O. Papaemmanouil and E. Upfal. "Performance Prediction for Concurrent Database Workloads". In *Proc. of SIGMOD'11.* Athens, Greece. June 2011. pp. 337-348.

[19] S. Elnaffar, P. Martin and R. Horman. "Automatically Classifying Database Workloads". In *Proc. of CIKM'02.* McLean, VA, USA. November 4-9, 2002. pp. 622-624.

[20] D. F. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini. "Economic Models for Allocating Resources in Computer Systems". In *Market-based Control: A Paradigm for Distributed Resource Allocation.* World Scientific Publishing Co., Inc. River Edge, NJ, USA, 1996. pp. 156-183.

[21] A. Ganapathi, H. Kuno, U. Dayal, J. Wiener, A. Fox, M. Jordan and D. Patterson. "Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning". In *Proc. of ICDE '09.* March 29 - April 2, 2009. Shanghai, China. pp. 592-603.

[22] A.G. Ganek and T.A. Corbi, "The Dawning of the Autonomic Computing Era". In *IBM System Journal,* Vol. 42, Issue 1, Jan. 2003. pp. 5-18.

[23] C. Gupta, A. Mehta and U. Dayal. "PQR: Predicting Query Execution Times for Autonomous Workload Management". In *Proc. of the 5th Intl. Conf. on Autonomic Computing (ICAC'08).* Chicago, IL, USA. June 2-6, 2008. pp. 13-22.

[24] C. Gupta, A. Mehta, S. Wang, and U. Dayal. "Fair, Effective, Efficient and Differentiated Scheduling in an Enterprise Data Warehouse". In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology,* Saint Petersburg, Russia, 2009. pp. 696-707.

[25] G. Graefe. "Query Evaluation Techniques for Large Databases". *ACM Computing Surveys.* Vol. 25, Issue 2, June 1993. pp. 73-169.

[26] H. Heiss and R. Wagner. "Adaptive Load Control in Transaction Processing Systems". In *Proc. of the 17th Intl. Conf. on Very Large Data Bases (VLDB'91).* pp. 47-54.

[27] J. M. Hellerstein, M. Stonebraker and J. Hamilton. "Architecture of a Database System". *Foundations and Trends in Database.* Volume 1, Issue 2, 2007. pp. 141-259.

[28] J. L. Hellerstein, Y. Diao, S. Parekh and D. M. Tilbury, "Feedback Control of Computing Systems", *IEEE Press,* Wiley-Interscience, John Wiley & Sons, Inc, 2004.

[29] M. C. Huebscher and J. A. McCann. "A Survey of Autonomic Computing - Degrees, Models, and Applications". *ACM Computing Surveys,* Volume 40, Issue 3, August 2008, Article No. 7.

[30] IBM Corp., "IBM DB2 Database for Linux, UNIX, and Windows Documentation". *On-line Documents.* https://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp

[31] IBM Corp., "Cognos Business Intelligence 10.2.2 Product Documentation". *On-line Documents.* http://www-01.ibm.com/support/docview.wss?uid=swg27042003

[32] J. O. Kephart and D. Chess, "The Vision of Autonomic Computing". *Computer,* Volume 36, Issue 1, January 2003. pp. 41-50.

[33] J. O. Kephart, "Research Challenges of Autonomic Computing". In *Proceedings of the 27th International Conference on Software Engineering,* St. Louis, MO, USA, 2005. pp. 15-22.

[34] J. O. Kephart and R. Das, "Achieving Self-Management via Utility Functions," *IEEE Internet Computing*. IEEE Educational Activities Department, Piscataway, NJ, USA. Vol. 11, Issue 1, Jan. 2007, pp. 40-48.

[35] L. Kleinrock. "Queueing Systems: Volume 1, Theory". John Wiley & Sons, 1975.

[36] D. Kossmann, "The State of the Art in Distributed Query Processing". In *ACM Computing Surveys*, Volume 32, Issue 4, December 2000. pp. 422-469.

[37] S. Krompass, H. Kuno, J. L. Wiener, K. Wilkison, U. Dayal and A. Kemper, "Managing Long-Running Queries". In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, Saint Petersburg, Russia, 2009. pp. 132-143.

[38] S. Krompass, A. Scholz, M. C. Albutiu, H. Kuno, J. Wiener, U. Dayal and A. Kemper. "Quality of Service-Enabled Management of Database Workloads". In *Special Issue of IEEE Data Engineering Bulletin on Testing and Tuning of Database Systems,* IEEE Computer Society, 2008.

[39] S. Krompass, H. Kuno, U. Dayal and A. Kemper, "Dynamic Workload Management for Very Large Data Warehouses - Juggling Feathers and

Bowling Balls", In *Proceedings of 33rd International Conference on Very Large Databases*. Vienna, Austria, 2007. pp. 1105-1115.

[40] E. Lazowska, J. Zahorjan, G. S. Graham and K. C. Sevcik "Quantitative System Performance: Computer System Analysis Using Queueing Network Models". Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1984.

[41] K. Lee *et al.* "Operator and Query Progress Estimation in Microsoft SQL Server Live Query Statistics". In *Proc. of SIGMOD'16*. San Francisco, USA, June 26 – July 1, 2016.

[42] J. Li, A. C. König, V. Narasayya and S. Chaudhuri. "Robust Estimation of Resource Consumption for SQL Queries Using Statistical Techniques". In *Proc. of the VLDB Endowment*. Vol. 5 Issue 11, July 2012. pp. 1555-1566.

[43] J. Li, R. V. Nehme and J. Naughton. "GSLPI: A Cost-Based Query Progress Indicator". In *Proc. of the 28th Intl. Conf. on Data Engineering (ICDE '12)*. Washington, DC, USA. 2012. pp. 678-689.

[44] J. Lobo, R. Bhatia and S. Naqvi, "A Policy Description Language", in *Proc. of 16th National Conf. on Artificial Intelligence, AAAI/IAAI,* Orlando, Florida, USA, 1999. pp. 291-298.

[45] G. Luo, J. F. Naughton, C. J. Ellmann, and M. W. Watzke. "Increasing the Accuracy and Coverage of SQL Progress Indicators". In *Proc. of the 21st Intl. Conf. on Data Engineering (ICDE '05)*. Tokyo, Japan. April 5-8, 2005. pp. 853-864.

[46] P. Martin, M. Zhang, W. Powley, H. Boughton, P. Bird and R. Horman. "The Use of Economic Models to Capture Importance Policy for Autonomic Database Management Systems". In *Proc of the 1st ACM/IEEE Workshop on Autonomic Computing in Economics in Conjunction with the 8th Intl. Conf. on Autonomic Computing (ICAC'11)*. 2011. Germany. pp. 3-10.

[47] A. Mehta, C. Gupta and U. Dayal. "BI Batch Manager: A System for Managing Batch Workloads on Enterprise Data-Warehouses". In *Proc. of the 11th Intl. Conf. on Extending Database Technology: Advances in Database Technology (EDBT'08)*. Nantes, France. March 25-30, 2008. pp. 640-651.

[48] A. Mehta, C. Gupta, S. Wang and U. Dayal, "Automated Workload Management for Enterprise Data Warehouses". In *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering,* Vol.31, No.1, March 2008. pp. 11-19.

[49] D.A. Menascé and J. O. Kephart, "Guest Editors' Introduction: Autonomic Computing". In *IEEE Internet Computing,* Vol. 11, Issue 1, January 2007. pp. 18-21.

[50] Microsoft Corp., "Managing SQL Server Workloads with Resource Governor". http://msdn.microsoft.com/en-us/library/bb933866.aspx

[51] Microsoft Corp., "Query Governor Cost Limit Option". http://msdn.microsoft.com/en-us/library/ms190419.aspx

[52] Microsoft Corp., "In-Memory OLTP". https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp

[53] MemSQL Inc., *Online Docs*. http://www.memsql.com/

[54] Y. Meng, P. Bird, P. Martin and W. Powley, "An approach to managing the execution of large SQL queries". In CASCON '07: *Proc. of the 2007 Conf. of the Center for Advanced Studies on Collaborative Research*. Toronto, Canada. October 2007. pp. 268 - 271.

[55] C. Mishra, N. Koudas. "The design of a query monitoring system". *ACM Trans. Database Syst.,* 34(1), 2009.

[56] A. Moenkeberg and G. Weikum, "Performance Evaluation of an Adaptive and Robust Load Control Method for the Avoidance of Data Contention Thrashing". In *Proc of the 18th Intl. Conf. on Very Large Data Bases (VLDB'92)*, Vancouver, BC, Canada, pp. 432- 443.

[57] J. Moffett and M. Sloman. "Policy Hierarchies for Distributed Systems Management". In *IEEE Journal on Selected Areas in Communications,* Volume 11, Issue 9, December, 1993.

[58] D. Narayanan, E. Thereska and A. Ailamaki. "Continuous Resource Monitoring for Self-Predicting DBMS". In *Proc. of the 13th IEEE Intl. Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. Atlanta, Georgia, USA. September 27-29, 2005. pp. 239-248.

[59] B. Niu, P. Martin and W. Powley, "Towards Autonomic Workload Management in DBMSs". In *Journal of Database Management*, 20(3), 1-17, July-September 2009.

[60] 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*. Toronto, Canada. October 2006.

[61] Oracle Corp., "Oracle Database Resource Manager". *On-line Documents*. http://download.oracle.com/docs/cd/B28359_01/server.111/b28310/dbrm.htm#i1010776

[62] Oracle Corp., "Oracle Database In-Memory with Oracle Database 12c Release 2". *An Oracle White Paper*, March 2017.

[63] Pivotal Greenplum. "Pivotal Greenplum Database v4.2.8 Documentation". http://gpdb.docs.pivotal.io/gpdb-428.html

[64] S. Parekh, K. Rose, J. Hellerstein, S. Lightstone, M. Huras and V. Chang. "Managing the Performance Impact of Administrative Utilities". In *Proc. of Self-Managing Distributed Systems,* Springer Berlin, Heidelberg, February 2004. pp. 130-142.

[65] W. Powley, P. Martin, M. Zhang, P. Bird and K. McDonald. "Autonomic Workload Execution Control Using Throttling". In *Proc. of 2010 IEEE 26th International Conference on Data Engineering Workshops (5th International Workshop on Self-Managing Database Systems),* Long Beach, CA, USA. March 1-6, 2010.

[66] W. Powley, P. Martin and P. Bird, "DBMS Workload Control Using Throttling: Experimental Insights". In CASCON '08: *Proc. of the 2008 Conf. of the Center for Advanced Studies on Collaborative Research*. Toronto, Canada. October 2008. pp. 1-13.

[67] R. Ramakrishnan and J. Gehrke. "Database Management Systems" (3rd Edition). McGraw-Hill. 2003.

[68] SAP HANA, *Online Docs*. https://www.sap.com/products/hana.html

[69] B. Schroeder, M. Harchol-Balter, A. Iyengar, E. Nahum and A. Wierman. "How to Determine a Good Multi-Programming Level for External Scheduling". In *Proc. of the 22nd Intl. Conf. on Data Engineering*. Atlanta, GA, USA. April 3-8, 2006.

[70] B. Schroeder, A. Wierman and M. Harchol-Balter. "Open vs. Closed: A Cautionary Tale". In *Proc. of the 3rd Conf. on Networked Systems Design & Implementation (NSDI'06)*. 2006.

[71] Teradata Corp., "Teradata Workload Analyzer User Guide". http://www.info.teradata.com/download.cfm?ItemID=1002974

[72] Teradata Corp., "Teradata Dynamic Workload Manager User Guide". http://www.info.teradata.com/download.cfm?ItemID=1005768

[73] Q. T. Tran, K. Morfonios and N. Polyzotis. "Oracle Workload Intelligence". In *Proc. of SIGMOD'15*. Melbourne, Victoria, Australia. May 31 - June 04, 2015. pp. 1669-1681.

[74] R. Vaupel, "z/OS Workload Manager: How It Works and How to Use It", *a White Paper of IBM*. April 2014.

[75] W.E. Walsh, G. Tesauro, J.O. Kephart, R. Das. "Utility Functions in Autonomic Systems", In *Proc. of the 1st Intl. Conf. on Autonomic Computing*. New York, USA. May 17-18, 2004. pp. 70-77.

[76] T. Wasserman, P. Martin and H. Rizvi, "Sizing DB2 UDB Servers for Business Intelligence Workloads". In *Proc. of the Conf. of the Center for*

Advanced Studies on Collaborative Research (CASCON'04). Toronto, Canada. October, 2004. pp. 135-149.

[77] G. Weikum, A. Moenkeberg, C. Hasse and P. Zabback. "Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering". In *Proceedings of 28th International Conference on Very Large Databases,* Hong Kong, China. August 20-23, 2002. pp. 20–31.

[78] M. Zhang, P. Martin, W. Powley and P. Bird. "Using Economic Models to Allocate Resources in Database Management Systems", In CASCON '08: *Proc. of the 2008 Conf. of the Center for Advanced Studies on Collaborative Research.* Toronto, Canada. October 2008. pp. 248-259.

[79] M. Zhang, P. Martin, W. Powley, P. Bird, and K. McDonald. "Discovering Indicators for Congestion in DBMSs". In *Proc. of the 7th Intl. Workshop on Self-managing Database Systems in Conjunction with the 28th Intl. Conf. on Data Engineering (ICDE'12).* Washington, DC, USA. 2012. pp. 263-268.

[80] M. Zhang, P. Martin, W. Powley, P. Bird and D. Kalmuk. "A Framework for Autonomic Workload Management in DBMSs". *it - Information Technology.* Volume 56, Issue 1, Feb 2014.

Mingyi Zhang is a Database Kernel Engineer at Huawei America Research in Santa Clara, California. He holds a PhD in Computer Science from Queen's University. His research interests include performance management in database management systems, cloud computing and autonomic computing.

Patrick Martin is a Professor in the School of Computing at Queen's University and the Director of the Database Systems Laboratory. He is a faculty fellow and a Visiting Scientist at the IBM's Centre for Advanced Studies and a Scotiabank Scholar. His research interests include big data analytics, database system performance, cloud computing and autonomic computing systems.

Wendy Powley is a Lecturer in the School of Computing at Queen's University. She holds a Masters of Science in Computer Science from Queen's University and worked as a Research Associate in the Database Systems Laboratory from 1992–2016. Her research interests include autonomic computing, workload management for database management systems and cloud computing.

Jianjun Chen is a Tech VP and head of advanced database research and development group in Huawei US Santa Clara R&D Center. He has 15+ year working experience in database area. He had been working on many database related products in Microsoft, Yahoo lab!, and Google before joining Huawei. He graduated with a Ph.D from Computer Science department of University of Wisconsin, Madison in 2002.