Utility Function-based Workload Management for DBMSs

Mingyi Zhang[†], Baoning Niu[§], Patrick Martin[†], Wendy Powley[†]
[†]Queen's University, Kingston, ON, Canada
[§]Taiyuan University of Technology, China
{myzhang, niu, martin, wendy}@cs.queensu.ca

Paul Bird, Keith McDonald
Toronto Software Lab, IBM Canada Ltd.
Markham, ON, Canada
{pbird, kmcdonal}@ca.ibm.com

***Abstract*—In the practice of autonomic computing, utility functions have been applied for achieving self-optimization in autonomic computing systems. Utility functions are suggested to guide an autonomic computing system to optimize its own behavior in accordance with high-level objectives specified by the system administrators. In this paper, we present two concrete examples to illustrate how utility functions can be properly defined in database workload management systems. Our utility functions help the systems translate high-level workload business importance policies into low-level policies of database system tuning actions, and therefore ensure the workloads achieve their required performance objectives. We also discuss the fundamental properties of a proper utility function that can be used for building autonomic workload management for database management systems.**

Keywords-Workload Management; Database Management Systems; Utility Functions; Autonomic Computing

I. INTRODUCTION

A database workload is a set of requests that have some common characteristics such as application, source of request, type of query, priority, and performance objectives (*e.g.*, response time goals) [3]. Workload management in database management systems (DBMSs) is a performance management process whose primary objective is to minimize the response time of critical queries, such as the ones for directly generating revenue for business organizations, or those issued by a CEO or VP of the organizations, maintain DBMSs running in an optimal state (*i.e.*, neither under-utilized nor overloaded), and maximize throughput of the system under these constraints. For both strategic and financial reasons, business organizations are consolidating individual database servers onto a shared data server to serve as the single source of corporate data. As a result, multiple types of requests are mixed and present on a single data server simultaneously. Request types include on-line transaction processing (OLTP) requests that are typically short and efficient, consume minimal system resources, and complete in sub-seconds, as well as business intelligence (BI) queries that can be longer, more complex and resource-intensive, and may require hours to complete. Requests generated by different applications or initiated from different business units may have unique performance objectives that are normally expressed in terms of service level agreements that must be strictly satisfied for business success.

Multiple requests running on a data server inevitably compete for shared system resources, such as system CPU cycles, buffer pools in main memory, disk I/O, and various queues in the database system. If some requests, for example, long BI queries, are allowed to consume a large amount of system resources without control, the concurrently running requests may have to wait for the queries to complete and release their used resources, thereby resulting in the waiting requests missing their performance objectives and the entire data server suffering degradation in performance. Moreover, requests presenting on a data server can fluctuate rapidly among multiple types, so it becomes impossible for database administrators to manually adjust the system configurations in order to dynamically achieve performance objectives of the requests during runtime. Therefore, a*utonomic* workload management becomes necessary and critical to effectively control the process of requests and manage shared system resources to achieve required performance objectives in a complex request mix environment.

Since *autonomic computing* was first introduced [4], a great deal of effort has been put forth by researchers and engineers in both academia and industry to build autonomic computing systems. Autonomic computing systems (with the ability to self- configure, optimize, protect, and heal) are self-managing systems that manage their own behavior in accordance with high-level objectives specified by human administrators [4], [6]. Such systems regulate and maintain themselves without human intervention to reduce the complexity of system management and dynamically achieve system objectives, such as performance, availability and security objectives. In particular, an autonomic workload management system for DBMSs is a self-managing system that dynamically manages workloads present on a data server in accordance with specified high-level business objectives.

Achieving the goal of autonomic workload management involves using utility functions, a measure of preference of a choice, to map high-level business objectives to low-level DBMS tuning actions in order to guide a database system to optimize its own behavior and achieve required performance objectives. In this paper, we illustrate the use of utility functions in different aspects of workload management, namely dynamic resource allocation and query scheduling, to ensure mixed requests on a data server achieve their required performance objectives, and present a set of fundamental properties of a utility function used for building autonomic workload management systems. The primary objective of our research is to explore autonomic workload management for DBMSs. The contribution of this study is a method of evaluating utility functions in workload management systems for DBMSs as a step towards a self-managing system.

The paper is organized as follows. Section II reviews the background and related work of this study. Section III discusses fundamental properties of a utility function that can be used in autonomic workload management for DBMSs. Section IV presents two different types of utility functions defined in our studies. Finally, we conclude our work and propose future research in Section V.

II. BACKGOUND

In the past several years, considerable progress has been made in workload management for DBMSs. New techniques have been proposed by researchers, and new features of workload management facilities have been implemented in commercial DBMSs. These workload management facilities include IBM® DB2® Workload Manager and Query Patroller [5], Teradata® Active System Management [17], Microsoft® SQL Server Resource and Query Governor [11], [12] and Oracle® Database Resource Manager [16]. The workload management facilities manage complex workloads (that is, a mix of business processing and analyzing requests) present on a data server using predefined database procedures, based on the characteristics of the requests, such as estimated costs, resource demands and performance behavior. The predefined procedures use workload characterization, admission control, and execution control mechanisms to impose proper controls on the workloads to achieve their performance objectives.

Krompass *et al.* [9] and Metha *et al.* [13] presented a process of workload management for DBMSs that included three typical controls, namely admission, scheduling, and execution controls. Admission control determines whether or not an arriving request can be admitted into a database system, thus it can avoid increasing the load while the system is busy. Query scheduling determines the execution order of admitted queries and maintains the database system running in an optimal state. Execution control dynamically manages some running queries to limit their impact on other concurrently running requests.

Autonomic computing has been intensively studied in the past decade. Many autonomic computing components (with certain self-managing capabilities) have been developed and proven to be useful in their own right, although a large-scale fully autonomic computing system has not yet been realized [7], [14]. In particular, Tesauro *et al.* [18], [20] studied autonomic resource allocation among multiple applications based on optimizing the sum of utility for each application. In their work, a data center was used, which consisted of multiple and logically separated application environments (AEs). Each AE provided a distinct application service using a dedicated, but dynamically allocated, pool of servers, and each AE had its own service-level utility function specifying the utility to the data center from the environment as a function of some service metrics. The authors presented two methodologies, a queuing-theoretic performance model and model-free reinforcement learning, for estimating the utility of resources. They evaluated the two methodologies in the data center and highlighted tradeoffs between the two methods. Bennani *et al.* [2] presented another approach for the same resource allocation problems in the autonomic data center. In their approach, the predictive multi-class queuing network models were proposed to implement the service-level utility functions for each AE.

III. UTILITY FUNCTIONS IN WORKLOAD MANAGEMENT

Achieving autonomic workload management for DBMSs involves the use of utility functions. In this section we consider the following questions:

- Why are utility functions suited to autonomic workload management for DBMSs?
- What utility functions are good for building autonomic workload management systems?

The first question can be answered based on the research of Kephart *et al.* [8] and Walsh *et al.* [20], who proposed the use of utility functions to achieve self-managing systems. In the work, the authors presented utility functions as a general, principled and pragmatic way of representing and managing high-level objectives to guide the behavior of an autonomic computing system. Two types of policies were discussed in guiding behavior of a system, namely action policies and goal policies. An action policy is a low-level policy that is represented in the form of *IF (conditions) THEN (actions*). Namely, if some conditions are satisfied, then certain actions must be taken by the system. In contrast with an action policy, a goal policy only expresses high-level objectives of a system, and the system translates the high-level objectives into specific actions to every possible condition. Utility functions are proposed for the translation as it has the property of mapping system states to real numbers, and the largest number represents a system preferred state. In using utility functions, a computing system, via maximizing its utilities under each condition, recognizes what the goal states are, and then decides what actions it needs to take in order to reach those states. Thus by maximizing utilities, a computing system optimizes its own behavior and achieves the specified high-level objectives.

As introduced in Section I, in a mixed request data server environment, the concurrently running workloads can have different types, business importance levels, and performance objectives. These properties may dynamically change during runtime. It is impossible for human administrators to manually make an optimal resource allocation plan for all the workloads in order to meet their resource requirements, whereas, a utility function is suited for this situation, based on the properties discussed above. It dynamically identifies resource preferences for a workload during runtime, and all utility functions of the workloads can be further used to define an objective function. A solution to optimizing the objective function is an optimal resource allocation plan. Autonomic workload management systems use the resource allocation plan to allocate resources to the workloads and achieve the required performance objectives. Thus, to manage workloads in DBMSs, using utility functions is naturally a good choice.

To answer the second question, we start by discussing performance behavior of a workload. The performance of a running workload on a data server depends on the amount of desired system resources that the workload can access. Typically, the performance of a workload increases non-linearly with the amount of resources assigned to it. As an example, in executing an OLTP workload, by increasing the multi-programming levels, the throughput of the workload initially increases, but at a certain point the throughput starts to level off. That is, at the beginning when the workload starts to run with a certain amount of resource allocated, performance of the workload increases rapidly. However, with additional resources assigned to the workload, the performance increment of the workload becomes very small. This can be caused either by a bottleneck resource among the system resources, which significantly limits the workload performance increase, or it may be the case that the database system has become saturated.

Utility functions in database workload management must capture the performance characteristics of a workload and represent the trend of the changes in amount of assigned resources and the achieved performance. A utility function defined for database workload management should be a monotonically non-decreasing function, and is capable of mapping the performance achieved by a workload with a certain amount of allocated resources into a real number, *u*. There is no single way to define a utility function, and with certain practice or research requirements, utility functions can have widely varying properties. However, we believe the following properties are necessary for a proper utility function used in workload management for DBMSs:

- The value, *u*, should increase as the performance of a workload increases, and decreases otherwise.
- The amount of utility increase (*i.e.*, marginal utility) should be large as the performance of a workload increases quickly, while the marginal utility should be small as the performance of a workload increases slowly.
- The input of a utility function should be the amount of resources allocated to a workload or a function of the allocated resources, and the output, *u*, should be a real number without any unit.
- The value, *u*, should not increase as more resources are allocated to a workload when the workload has reached its performance objective.
- In allocating multiple resources to a workload, a utility function should be capable of identifying the critical resources for the workload.
- For objective function optimization, a utility function should have good mathematical properties, such as an existing second derivative.

The first two properties describe the general performance behavior of a workload, and the third property provides the domain and codomain for a utility function. These three properties are fundamental for a utility function that can be used in a workload management system for DBMSs. The fourth and fifth properties represent the relationships among workload performance, resource provisions and performance objectives. Namely, if a workload has met the performance objective, the utility function would inform the database system not to allocate more resources to the workload, and if there is a critical resource for a workload, the utility function would tell the system to provide the resource. The last property presents a way of effectively optimizing objective functions.

IV. Utility Function Applications

Two examples of the use of utility functions in workload management for DBMSs are presented in this section, namely dynamic resource allocation and query scheduling. The two utility functions are discussed with respect to the properties listed in Section III.

A. Dynamic Resource Allocation

In workload management for DBMSs, dynamic resource allocation can be triggered by workload reprioritization (a workload execution control approach [5], [21]). That means, a workload's business priority may be dynamically adjusted as it runs, thereby resulting in immediate resource reallocation to the workload according to the new priority.

Two shared resources are considered in the study, namely database buffer pool memory pages and system CPU shares. The DBMS concurrently runs multiple workloads, which are classified into different business importance classes with unique performance objectives. High importance workloads are assigned more resources, while low importance ones are assigned fewer. In the approach, all the workloads are assigned some virtual "wealth" to reflect the business importance levels. High importance workloads are assigned more wealth than low importance ones.

The dynamic resource allocation approach consists of three main components, namely a *resource model*, a *resource allocation method* and a *performance model*. The *resource model* is used to partition the resources and to determine a reasonable total amount of the resources for allocation. The *resource allocation method* determines how to obtain an optimal resource pair of buffer pool memory pages and CPU shares for a workload. In the approach, a greedy algorithm is used for identifying resource preferences of a workload. The resource allocation is determined iteratively. In an iteration of the algorithm, by using its virtual wealth, a workload bids for a unit of the resource (either buffer pool memory pages or CPU shares) that it predicts will yield the greatest benefit to its performance. The *performance model* predicts the performance of a workload with certain amount of allocated resources in order to determine the benefit of the resources. In the approach, queueing network models (QNM) [10] are used to predict the performance of a workload at each step of the algorithm. The details of this study are available elsewhere [21].

We consider OLTP workloads and use throughput as the performance metric to represent the performance required and achieved by the workloads. We model the DBMS used in our experiments for each workload with a single-class closed QNM, which consists of a CPU service center and an I/O service center. The CPU service center represents the system CPU resources and the I/O service center represents buffer pool and disk I/O resources. The request concurrency level of a workload in the DBMS is the number of database agents (CPU resources) assigned to the workload. The average CPU service demand of requests in the workload can be expressed as a function of the CPU shares allocated to the workload, using equation (1).

$$S_{CPU} = 1/(a * N + d) \tag{1}$$

We experimentally defined the relationship between the CPU service demand and the number of database agents used in a DBMS. In the equation, *N* is database agents and $N \leq m$, and a and d as well as *m* are constants that can be determined through experimentation.

For an OLTP workload, the average I/O service demand can be expressed as a function of buffer pool memory size, which can be derived from *Belady's* equation [1]. The I/O service demand is:

$$S_{IO} = c * M^b \tag{2}$$

where *c* and *b* are constants, and *M* is buffer pool memory pages assigned to the workload. In the equation the constants *c* and *b* can be determined through experimentation.

Performance of a workload with some allocated resources, <*cpu, mem*>, can be predicted by solving the analytical performance model with Mean Value Analysis (MVA) [10]. The predicted throughput of a workload can be expressed as a function of its allocated resources, using equation (3).

$$X = MVA(N,\ S_{CPU}(cpu),\ S_{IO}(mem),\ Z) \tag{3}$$

where, *X* is the predicted throughput of a workload by using MVA on the QNM for a workload with its allocated resource pair, <*cpu, mem*>; *N* is the number of requests concurrently running in the system (*i.e.*, the number of database agents assigned to the workload); $S_{CPU}(cpu)$ is the average CPU service demand determined in equation (1); $S_{IO}(mem)$ is the average I/O service demand determined in equation (2); and *Z* is think time.

To guide workloads to capture appropriate resource pairs, utility functions are employed in the approach. We define a utility function that normalizes the predicted throughput from our performance model relative to the maximum throughput that the workload could achieve when all the resources are allocated to it. The utility function is given by:

$$u = MVA_{throughput}(N,\ S_{CPU}(cpu),\ S_{IO}(mem),\ Z)/X_{max} \tag{4}$$

where, $MVA_{throughput}(w,\ x,\ y,\ z)$ is the predicted throughput determined in equation (3), and X_{max} is the maximum throughput achieved by a workload with all the resources allocated, which can be determined through experimentation.

This utility function is defined to map performance achieved by a workload with certain amount of resources into a real number *u*, $u \in [0, \ldots, 1]$. If the utility of resources allocated to a workload is close to 1, it means the performance of the workload is high, while if the utility of resources allocated to a workload is close to 0, it means the performance of the workload is low. Workloads employ the utility function to calculate marginal utilities, that is, the difference in utilities between two possible consecutive resource allocations in a resource allocation process. As the utility function is non-decreasing, the value of a marginal utility is also in the range [0, …, 1].

The marginal utility reflects potential performance improvement of a workload. For some resources, if the calculated marginal utility of a workload is close to 1, then it means these additional resources can significantly benefit the workload's performance, while if the calculated marginal utility is close to 0, then the additional resources will not greatly improve the workload's performance. By examining the marginal utility value, a workload can determine the preferred resources for bid. The bid of a workload is the marginal utility multiplied by current available wealth of the workload, and indicates that a workload is willing to spend the marginal-utility percentage of its current wealth as a bid to purchase the resources. Wealthy workloads, therefore, can acquire more resources in the resource allocation processes and achieve better performance than poor ones.

B. Query Scheduling

Our *query scheduler* [14] is built on a DB2 DBMS and uses DB2 Query Patroller (DB2 QP) [5], a query management facility, to intercept arriving queries and acquire their information, and then determines an execution order of the queries. The *query scheduler* works in two main processes, namely workload detection and workload control. The workload detection process classifies arriving queries based on their service level objectives (SLOs), and the workload control process periodically generates new plans to respond to the changes in the mix of arriving requests.

We consider a system with *n* service classes, each with a performance goal and a business importance level, denoted as $\langle \overline{g}_i, m_i \rangle$, where $\overline{g}_i$ is the performance goal of the *i-th* service class, and m_i is the class business importance level. The pair $\langle \overline{g}_i, m_i \rangle$ is a service level objective. We denote $g_1, g_2, \cdots, g_n$ as the predicted performance of the *n* service classes given a resource allocation plan $r_1, r_2, \cdots, r_n$ (multi-programming levels in our case). The performance of the *i-th* service class, g_i, can be predicted by using a performance model given r_i, the amount of resources allocated to the service class. The utility of the *i-th* service class, u_i, can be expressed as a function of $\overline{g}_i$, m_i and g_i, namely $u_i = f_i(\overline{g}_i, m_i, g_i)$, and the *n* SLOs can be encapsulated into an objective function *f*, $f(u_1, u_2, \cdots, u_n)$. Thus, the scheduling problem can be solved by optimizing the objective function *f*. By properly defining the functions of f_i 's and *f*, an optimal resource allocation plan can be derived by maximizing the objective function.

In the study, we specifically consider business analysis requests, such as those found in data warehousing decision support systems. The TPC-H benchmark [19] is used as the database and workloads in our experiments. We apply the performance metric - *query execution velocity*, the ratio of expected execution time of a query to the actual time the query spent in the system (the total time of execution and delay), to represent the performance required and achieved by the queries, since queries in decision support systems can widely vary in their response times. In using query execution velocity, performance goals as well as business importance levels of queries can be well captured.

Based on the experiments we found the following general form of utility functions satisfies our requirements:

$$u = 1 - e^{\alpha m \frac{\bar{g}-g}{g-\hat{g}}} \tag{5}$$

where, $\bar{g}$ is the performance goal of a service class to be achieved, *m* is the importance level of the service class, $\hat{g}$ is the lowest performance allowed for the service class, *g* is the actual performance, and α is an importance factor that is a constant and can be experimentally determined or adjusted to reflect the distance between two adjacent importance levels. In using α, we control the size and shape of the utility function.

The objective function, *f*, is then defined as a sum of the service class utility functions, using equation (6):

$$f = \sum_{i=1}^{n} u_i \tag{6}$$

In *query scheduler*, the *performance solver* employs a performance model to predict query execution velocity for a service class. That is, given a new value of service class cost limit, the performance of the service class can be predicted for the next control interval, which is based on its performance and service class cost limit at the current control interval. The performance at the next control interval is predicted by:

$$V_i^k = \begin{cases} V_i^{k-1} C_i^k / C_i^{k-1} & if\, V_i^{k-1} C_i^k / C_i^{k-1} \le 1 \\ 1 & if\; V_i^{k-1} C_i^k / C_i^{k-1} > 1 \end{cases} \tag{7}$$

where, V_i^{k-1} and V_i^k are query execution velocity of service class *i* at *(k-1)-th* and *k-th* control intervals, respectively; C_i^{k-1} and C_i^k are cost limits of service class *i* at the *(k-1)-th* and the *k-th* control intervals, respectively.

Therefore, a scheduling plan can be determined. From equations (5), (6) and (7), we have:

$$f = \sum_i^n u_i^k \tag{8}$$

$$u_i^k = 1 - e^{a_i m_i \frac{\bar{V}_i - V_i^k}{V_i^k - \hat{V}_i}} \tag{9}$$

$$V_i^k = V_i^{k-1} C_i^k / C_i^{k-1} \tag{10}$$

replacing V_i^k in equation (9) with equation (10) and u_i^k in equation (8) with equation (9), the solution for maximizing the objective function, $f(C_1^k, C_2^k, \cdots, C_n^k)$, is the query scheduling plan for *k-th* control interval, where the object function must maintain the constraint, $C_1^k + C_2^k + \cdots + C_n^k \le C$, and C is the system cost limits.

C. Discussion

In these two examples, for dynamic resource allocation, the utility function was defined based on a single-class closed QNM, while, for query scheduling, the utility function was chosen based on an exponential function. These two types of utility functions are different in their forms and research requirements, but both strictly maintain the same fundamental properties listed in Section III. The input of the dynamic resource allocation utility function is an amount of allocated resources (the resource pair, <*cpu*, *mem*>), the output is a real number in the range [0, 1], and the applied QNM properly predicts performance behavior of the workload. The input of the query scheduling utility functions is a performance model with allocated resources (the query execution velocity of a service class, as described in Subsection *B*) and the output is a real number in (-∞, +∞). Based on the exponential function properties, as the input of the utility function increases, the output (utility) increases and, at a certain value, it begins to level off. That means, when the service class approaches its performance goal, the utility increase is less, and it indicates that the database system should not assign more resources to the service class.

For a defined objective function, if it is continuous, the *Lagrange* method can be applied to solve it, otherwise searching techniques may be used. In query scheduling, the defined utility function has the second derivative existing, and this allows mathematical methods to be applied to optimize the objective function.

In evaluating the two types of utility functions based on the set of properties listed in Section III, both utility functions preserve the fundamental properties, that is, a) the utility increases as a workload performance increases, and decreases otherwise; b) the marginal utility is large as a workload performance increases quickly, and is small otherwise; c) the input and output are in the required types and values. In comparing the two utility functions presented in Table 1, we observe that the utility function used in dynamic resource allocation has the property of identifying critical resources for a workload, but it does not have mathematical properties for optimizing an objective function. The utility function used in query scheduling possesses a good mathematical property for optimizing its objective function, but it does not have the property of identifying system critical resources.

Since the utility functions were strictly defined based on their research requirements, the specific research problems shaped the utility function's properties. So, we conclude that the two types of utility functions are good in terms of their specific research requirements and considered acceptable based on the set of properties listed in Section III.

TABLE I. COMPARISION OF THE TWO UTILITY FUNCTIONS

	Utility functions in Dynamic Resource Allocation	**Utility functions in Query Scheduling**
Utility Increasing Normally	yes	yes
Marginal Utility Increasing Normally	yes	yes
Utility Function Input	allocated resources	a function of the allocated resources
Utility Function Output	a number in [0, 1]	a number in **(**-∞, +∞**)**
Critical Resource Identifying	yes	no
Having Mathematical Property	no	yes
Utility Increase Stops as Goals Achieved	yes	yes

V. Conclusion And Future Work

In this paper we have presented two concrete examples to illustrate how utility functions can be applied to database workload management, namely dynamic resource allocation and query scheduling. Based on the examples, we generalized a set of function properties that is fundamental for defining utility functions in building autonomic workload management for DBMSs in future practice and research.

As more workload management techniques are proposed and developed, we plan to investigate the use of utility functions to choose during runtime an appropriate workload management technique for a large-scale autonomic workload management system, which can contain multiple techniques. Thus, the system can decide what technique is most effective for a particular workload executing on the DBMS under certain particular circumstance.

Acknowledgment

This research is supported by IBM Centre for Advanced Studies (CAS), IBM Toronto Software Lab, IBM Canada Ltd., and Natural Science and Engineering Research Council (NSERC) of Canada.

References

[1] L. A. Belady. “A Study of Replacement Algorithms for a Virtual-Storage Computer”. IBM Systems Journal, Volume 5, Issue 2, June 1966, pp. 78-101.

[2] M. N. Bennani and D. A. Menasce, “Resource Allocation for Autonomic Data Centers using Analytic Performance Models”, In Proc. of the Intl. Conf. on Autonomic Computing, (ICAC’05), Seattle, Washington, USA, 13-16 June, 2005, pp. 229-240.

[3] D. P. Brown, A. Richards, R. Zeehandelaar and D. Galeazzi, “Teradata Active System Management: High-Level Architecture Overview”, A White Paper of Teradata, 2007.

[4] IBM Corp., “Autonomic Computing: IBM’s Perspective on the State of Information Technology”. On-line Documents, retrived in February 2011. http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf.

[5] IBM Corp., “IBM DB2 Database for Linux, UNIX, and Windows Information Center”. On-line Documents, retrived in Feb. 2011. https://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp

[6] J. O. Kephart and D. M. Chess, “The Vision of Autonomic Computing”, Computer, Volume 36, Issue 1, January 2003, pp. 41-50.

[7] J. O. Kephart, “Research Challenges of Autonomic Computing”. In Proc. of the 27th Intl. Conf. on Software Engineering (ICSE’05). St. Louis, MO, USA, 15-21 May, 2005, pp. 15-22.

[8] J. O. Kephart and R. Das, “Achieving Self-Management via Utility Functions,” IEEE Internet Computing, Vol. 11, Issue 1, January/February, 2007, pp. 40-48.

[9] S. Krompass, H. Kuno, J. L. Wiener, K. Wilkison, U. Dayal and A. Kemper, “Managing Long-Running Queries”, In Proc. of the 12th Intl. Conf. on Extending Database Technology: Advances in Database Technology (EDBT’09), Saint Petersburg, Russia, 2009, pp. 132-143.

[10] 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.

[11] Microsoft Corp., “Managing SQL Server Workloads with Resource Governor”. On-line Documents, retrived in February 2011. http://msdn.microsoft.com/en-us/library/bb933866.aspx.

[12] Microsoft Corp., “Query Governor Cost Limit Option”, On-line Documents, retrived in February 2011. http://msdn.microsoft.com/en-us/library/ms190419.aspx.

[13] 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.

[14] D. A. Menasce and J. O. Kephart, “Guest Editors' Introduction: Autonomic Computing”, In IEEE Internet Computing, Volume 11, Issue 1, January 2007, pp. 18-21.

[15] B. Niu, P. Martin and W. Powley, “Towards Autonomic Workload Management in DBMSs”, In Journal of Database Management, Volume 20, Issue 3, 2009, pp. 1-17.

[16] Oracle Corp., “Oracle Database Resource Manager”, On-line Documents, retrived in February 2011. http://download.oracle.com/docs/cd/B28359_01/server.111/b28310/dbrm.htm#i1010776.

[17] Teradata Corp., “Teradata Dynamic Workload Manager”, On-line Documents, retrived in February 2011. http://www.info.teradata.com/templates/eSrchResults.cfm?prodline=&txtpid=&txtrelno=&txtttlkywrd=tdwm&rdsort=Title&srtord=Asc&nm=Teradata+Dynamic+Workload+Manager.

[18] G. Tesauro, R. Das, W. E. Walsh and J. O. Kephart, "Utility-Function-Driven Resource Allocation in Autonomic Systems", In Proc. of the 2nd Intl. Conf. on Autonomic Computing (ICAC’05), Seattle, Washington, USA, 13-16 June, 2005, pp.342-343.

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

[20] W. E. Walsh, G. Tesauro, J. O. Kephart and R. Das. “Utility Functions in Autonomic Systems”, In Proc. of the 1st Intl. Conf. on Autonomic Computing (ICAC’04), New York, USA, 17-18 May, 2004, pp.70-77.

[21] M. Zhang, P. Martin, W. Powley and P. Bird. “Using Economic Models to Allocate Resources in Database Management Systems”, In Proc. of the 2008 Conf. of the Center for Advanced Studies on Collaborative Research (CASCON’08), Toronto, Canada, Oct. 2008, pp. 248-259.