Today's DBMSs: How *autonomic* are they?

Said Elnaffar, Wendy Powley, Darcy Benoit, and Pat Martin
{elnaffar, wendy, benoit, martin}@cs.queensu.ca

Technical Report 2003-469
School of Computing, Queen's University
Kingston, Ontario, Canada K7L 3N6

September 2003

Abstract

Database Management Systems (DBMSs) are complex systems whose manageability is increasingly becoming a real concern. Realizing that expert Database Administrators (DBAs) are scarce and that the cost of hiring them is a major part of the Total Cost of Ownership (TCO) makes an urgent call for an Autonomic DBMS (ADBMS) that is capable of managing and maintaining itself. In this paper, we examine the characteristics that a DBMS should have in order to be considered autonomic. We assess the position of today's DBMSs by drawing example features from popular, commercial database products, such as DB2 UDB, SQL Server, and Oracle. We argue that today's DBMSs are still far from being autonomic. We highlight the source of difficulties towards achieving that goal, and sketch the most important research terrains that need investigation in order to have ADBMSs one day.

1. Introduction

Database management systems (DBMSs) are a vital component of many mission-critical information systems and, as such, must provide high performance, high availability, excellent reliability and strong security. These DBMSs are managed by expert Database Administrators (DBAs) who must be knowledgeable in areas such as capacity planning, physical database design, systems tuning and systems management.

DBAs face increasingly more difficult challenges brought about by the growing complexity of DBMSs, which stems from several sources:

- **Increased emphasis on Quality of Services (QoS).** DBMSs are components of larger systems, such as electronic commerce applications, that support different levels of QoS depending on users' needs. A DBMS must provide service guarantees in order that the overall system can satisfy the end-to-end QoS requirements.
- **Advances in database functionality, connectivity, availability, and heterogeneity**. DBAs must grapple with complex decisions about hardware platforms, schema design, constraints and referential integrity, primary keys, indexes, materialized views, the allocation of tables to disks, and shared-nothing, shared-everything, or SMP-cluster topology.
- **Ongoing maintenance**. Once designed, databases require substantial human input to build, configure, test, tune, and operate. DBAs handle table reorganization, data statistics collection, backup control, security modeling and administration, disaster recovery planning, configuration and performance tuning, problem analysis, and more.

- **Burgeoning database size**. Data warehouses containing tens of terabytes of data are not uncommon. Popular applications such as SAP typically create more than 20,000 tables and support thousands of users simultaneously [4].
- **E-Service era**. The problems described above become more apparent where the internet presents to the DBMSs a broad diversity of workloads with high variability under sophisticated multi-tier architectures.

DBMS customers and vendors, because of the spiraling complexity, have recently begun to place an increased emphasis on reducing the Total Cost of Ownership (TCO) of systems. Despite the dramatic recent growth in database sizes, TCO is increasingly dominated by human costs, specifically the DBAs. A 1998 study by the Aberdeen Group [19] showed that a five-year, 25-user implementation of a leading industrial RDBMS incurred 81 percent of its TCO from the human costs of training, maintenance, and implementation. Similarly, a TCO report from D.H. Brown Associates [20] that compared two leading database products for both data warehouse and online transaction processing (OLTP) applications found that human costs represented a large component of TCO in all cases. Moreover, skilled DBAs and application developers are scarce.

Autonomic computing systems are a proposed approach to solving the above problems. An *autonomic computing system* has the following properties [27]:

- The system is "aware of itself" and able to act accordingly.
- The system is able to configure and reconfigure itself under varying and unpredictable conditions.
- The system is able to recover from events that cause it to malfunction.
- The system is able to anticipate optimized resources needed to perform a task.
- The system is able to protect itself.

We believe that Autonomic Database Management Systems (ADBMS) are a desirable long-term research goal. In pursuing this goal it is useful to evaluate current DBMSs in light of the properties of autonomic computing systems in order to judge what has been accomplished to date and what problems remain to be solved.

The goal of this paper is therefore to examine current DBMSs with respect to their embodiment of the concepts of autonomic computing systems. We focus on three popular DBMS products, namely IBM DB2 Universal Database Version 8.1 [29], Oracle 9i [24] and Microsoft SQL Server 2000 [26]. Our objective is to report on the current state of practice with respect to autonomic DBMSs based on a review of generally available materials such as research papers, white papers and system documentation. We provide examples, not an exhaustive list, of autonomic features. We do not attempt to draw comparisons between the DBMSs.

In examining the DBMSs, we believe that the autonomic features available in the systems can be identified as belonging to one of the following general categories, which correspond to the kinds of tasks that are typically performed by DBAs:

- **Plug-n-Play DBMSs.** These features support system set-up and initialization. They include initial capacity planning, DBMS installation, configuration, and deployment, and data migration.
- **Physical and Logical Design.** These features include support for tasks related to laying out the data on the storage devices and structuring them properly. Examples of such tasks are the selection of the most efficient indexes and materialized views, and partitioning tables [8].
- **Ongoing Preventive Maintenance.** This category encompasses features that aim to keep the system stable and performing satisfactorily. They support the phase in which the DBMS self-monitors in order to perform ongoing tasks such as self-tuning and self-reorganizing. Examples include support for defragmenting data and re-structuring indexes, creating backups, updating statistics, space management, user management, and table and object maintenance.
- **Problem Diagnosis and Correction.** These features help with identifying any anomalies in the system and determining their root cause, notifying the administrators, and taking corrective actions and tuning.
- **Availability and Disaster Recovery.** These features help the DBMS get back to its stable state or recover from a disaster. For example, the DBMS should be able to carefully analyze its log and identify the correct set of backup

assets it retains in order to get the system operational. They also support multiple server synchronization and maintenance.

The remainder of the paper is organized as follows. Section 2 presents our survey of the autonomic features of three popular commercial DBMSs. Section 3 summarizes the survey and points out further functionality required in DBMSs to achieve the goal of autonomic DBMSs. Section 4 summarizes the paper and presents our conclusions.

2. How autonomic are current DBMSs?

Ganek and Corbi identify important, general properties of an autonomic computing system [27]. In this section, we discuss the DBMS-specific autonomic characteristics and devote a sub-section to each characteristic in which we first detail what kind of automation a characteristic implies in the realm of DBMSs and then list some concrete examples drawn from commercial DBMSs that best match the description of the particular autonomic characteristic. We should note again that this is not meant to be an exhaustive list of features provided by the various DBMSs but instead we wish to outline where DBMSs are today in terms of autonomic capabilities.

2.1 Self-optimizing

Self-optimization is one of the most challenging features to include in a DBMS. It allows a DBMS to perform any task and execute any service utility in the most efficient manner given the present workload parameters, available resources, and environment settings. Obviously, the most important task in need of optimization is the execution of a query.

Since SQL statements are deemed the basic components of a DBMS workload, a remarkable effort has been devoted towards query optimization. In fact, optimizing queries is one of the most apparent autonomic features of today's DBMSs. In general, query optimization involves query translation (rewriting a query as a more efficient, semantically equivalent query), the generation of a cost-efficient execution plan and dynamic runtime optimizations [16][17][18][9].

The most complex task of query optimization is the choice of execution plan. Plans are generated based on cost models. In order to obtain more accurate models, the optimizer uses statistics and takes into account the column distributions to estimate the cardinality of the query. It is important for these statistics to be up to date and accurate, however this has traditionally be a disruptive process. Oracle [23] and SQL Server [21][22] provide facilities that automatically determine which columns require histograms and also which tables require new statistics. When the DBA issues the command to update the statistics, only those flagged by system will be updated. Oracle also supports a dynamic sampling feature that gathers statistics on the fly while a query is being optimized.

Query optimization can be time consuming and, for some queries, the time to optimize may not be worth the time spent on optimization. The DB2 optimizer allows the user to adjust the amount of optimization each query experiences. More sophisticated models, such as those found in Oracle [23] and SQL Server [26], automatically determine the appropriate amount of optimization on a per-query basis.

The majority of query optimizers adapt the produced cost models to the hardware settings such as the number and the speed of CPUs, network connecting machine clusters and the setup of storage devices. Therefore, the final cost models can be intricate [18] as they include factors such as the hit ratio, the various configurations of memory areas (multiple buffer pools, sort heaps, catalogue cache, etc.), the cost of building temporary tables versus re-scanning tables, non-uniformity of data distribution and pre-fetching.

During query execution, cost models will be able to benefit from the self-validation mechanism proposed by. DB2's Learning Optimizer (LEO) [10]. LEO is a promising, smart optimizer that learns from prior experiences by self-validating the cardinality model of queries, using the actual cardinalities measured by executing queries with similar predicates. This technology has not been commercialized yet but expected to be part of DB2's engine soon [15].

Dynamic adjustments to the execution strategy at, or during, runtime will result in more efficient use of hardware resources such as memory, disk and CPU, thus resulting in better overall performance. Oracle provides automatic memory allocation [23] so that each query has the appropriate amount of memory (within DBA specified limits). Memory usage may be adjusted even while the query is executing. DB2 and Oracle both provide an automatic query parallelism selection mechanism that determines at runtime when it is beneficial to employ parallel execution, and determines the most effective degree of query parallelism across SMP CPUs. Dynamic runtime optimizations not only

ensure optimal execution of an individual query but optimal execution in the context of all executing queries.

In addition to query optimization, a DBMS must also optimize the various utilities such as backup, restore, statistics collection and data load utilities to ensure that these jobs, when possible, are run during non-peak times and that they make the most efficient use of resources. DB2's Load utility, for example, automatically optimizes its performance after examining its environment settings. The Load utility performs mass insertions of data into a target table. It carries out its job efficiently by exploiting a series of parallel I/O sub-agents for pre-fetching, SMP parallelism degree, and the amount of memory available for buffering and sorting. Furthermore, the Load utility maintains the index of the processed table by making a non-trivial decision, depending on the complexity of the index data structure, of either rebuilding completely, or building the index incrementally as each data tuple is inserted.

The conditions of a database environment are ever changing and there will always be room for optimization. An autonomic DBMS will recognize this need, evaluate the current status and environment and take the necessary action. The DBMS must always be looking for ways to optimize overall system performance.

2.2 Self-configuring

The performance of a DBMS depends on the configuration of the hardware and software components. An autonomic DBMS should provide users with reasonable "out of the box" performance and dynamically adapt its configuration to provide acceptable, if not optimal, performance in light of constantly changing conditions. An ADBMS should recognize changes in its environment that warrant re-configuration, for example a workload change that places a new demand on the resources or the addition of new hardware, and it must react quickly with appropriate adjustments. It should also be able to reconfigure itself without severely disrupting online operations. A DBMS configuration includes performance parameters (or knobs), resource consumption thresholds, and the existence of auxiliary data structures such as indexes and materialized views in the database schema.

The ability to dynamically adjust DBMS tuning parameters without DBMS shut-down has only just emerged in the most recent versions of DB2, Oracle and SQL Server. For several tuning parameters, however, applications must disconnect before the new values take effect thus causing a disruption of service. Nonetheless, the ability to dynamically adjust some tuning parameters greatly increases the potential for self-configuration features in future DBMS releases.

Static "out of the box" configurations obviously cannot provide acceptable performance under all circumstances. Typically the configuration must be tailored to the application and the hardware environment. Therefore, DBMSs provide configuration wizards such as DB2's Configuration Advisor. This tool configures over 35 parameters pertaining to server agents, I/O subagents, logging, sorting, etc. Each parameter value is set in light of system characteristics such as total memory available, number of disks, and number of CPUs, and user-supplied information. A recent version of DB2's advisor demonstrated remarkable effectiveness in configuring the system for OLTP workloads [11].

Configuration advisors are tools to assist with initial configuration but the settings are, in most cases, static. The goal of an autonomic DBMS is to provide dynamic adjustment of these settings. Little support is provided for this type of self-configuration. SQL Server and Oracle both provide some degree of automatic memory management. These systems allocate memory as needed by the database, limiting memory allocation when either a user-imposed limit is reached or the system's physical resources run low.

Self-configuring features of an ADBMS should include support for determining the optimal set of indexes and materialized views to be used by the query optimizer. All the DBMSs provide an index advisor (DB2's Design Advisor [12][13], SQL Server's Index Wizard, and Oracle's Index Tuning Wizard) that recommends a suitable set of indexes. Recommendations are based on SQL statements that are either automatically captured from the DBMS or supplied by the user, as well as space constraints. Similar to the index advisor, SQL Server [14] and Oracle [25] also recommend the materialized views that the system can benefit from.

2.3 Self-healing

A fundamental requirement of a DBMS is that the database remains in, or can be restored to, a consistent state at all times. A DBMS must reliably log all operations, periodically archive the database and be able to use the logs and backups to recover from failure.

Ideally an ADBMS should recognize when a full or incremental backup is necessary and perform these operations with minimal system disruption. In the event of catastrophic failure, an ADBMS should be able to retrieve the most recent backup, restore to the consistent point just before the failure, then resume its halted operations after handling the exceptions.

All DBMSs support logging, backup and recovery mechanisms. Non-catastrophic failures (that is, those that can be repaired using only log files) are typically initiated automatically by the DBMS. The DBMS system can recognize that the database is in an inconsistent condition and use the log files to restore to a consistent state.

DB2 has a recovery tool, the *Recovery Expert*, which analyzes the recovery assets available and recommends a technique to be selected. For example, if a set of tables needs to be recovered to a point in time five minutes ago, Recovery Expert may recommend that the log be analyzed in order to generate UNDO SQL to effect the recovery. DB2's *Automatic Incremental Restore* mechanism uses the backup history for automatically searching for the correct backup images needed to complete the restore process successfully.

SQL Server and Oracle allow the DBA to set a recovery interval parameter that specifies a target for recovery time in seconds. The DBMS automatically adjusts the underlying logging and recovery systems in order to maintain the required recovery time.

Oracle provides automated facilities to ensure database backup integrity. A backup monitor assures that backups are performed as necessary in order to guarantee recovery.

The concept of self-healing also applies to the ability of the DBMS to correct problems that are interfering with good system performance or those that prevent an operation from completing. Oracle provides the ability to resume operations (such as a batch load) following corrective action (such as the addition of more disk space in the event that an "out of space" error occurs) [24].

2.4 Self-protecting

Database protection implies at least the following aspects: database security [5] [6], privacy **[7]**, analytical auditing mechanisms, and admission control strategies. These features shield the DBMS from potential, errant requests that may deteriorate its performance or bring the DBMS down.

All multi-user DBMSs provide authentication mechanisms that prevent unauthorized users from accessing the database. Of course, human intervention will always be required to determine those who should be granted access. Database privacy ensures that users are granted access only to the portions of the database that are required. DBMSs provide the ability to grant different types of access (select, insert, update, delete) to database objects. Current DBMSs differ in the level of access granularity; DB2 and SQL Server provide security on a per table basis whereas Oracle provides row-level security.

An ADBMS should provide *auditing* mechanisms where logs are used to track all DBMS activity. The DBMS can use this information to track trends, analyze potential threats, support future security planning, and assess the effectiveness of countermeasures. DBAs should define their auditing strategies based on their knowledge of the application or database activity around sensitive data, and the ADBMS should setup and take care of the detailed configuration.

Data encryption is a key element in protecting data as it adds an essential level of protection from intruders who break through firewalls or operating system and network features. It also deters malfeasance from internal users. The best solutions minimize performance impact by monitoring only the information that's critical from a security point of view instead of entire databases.

Admission and application control is essential for ADBMSs to protect the system from database requests that may deteriorate performance and/or undesirably consume system resources. DB2 provides mechanisms for controlling applications that are submitted for execution and those that are currently executing, based on the resources they consume. The first type of control is called a "predictive governor" because it uses the query optimizer's estimate of the relative resources each query is expected to consume to limit surges of arriving or long-running queries that could saturate the server. The second type is called a "reactive governor" because it monitors the actual resources consumed to prevent runaway queries from wasting resources.

The Oracle Resource Manager [24] provides automatic prioritization that detects long running operations and limits resource consumption so that other users do not experience delays. The proactive governor permits the ability to limit the number of concurrent long-running operations and prevents execution of resource intensive operations during peak periods. In addition, service level agreements can be

specified and monitored. A specified event is triggered if the service level agreement is violated.

2.5 Self-organizing

An ADBMS should be capable of dynamically re-organizing and re-structuring the layout of data stored in databases (e.g., tables), associated auxiliary data structures (e.g., indexes), and any system-related data (e.g., system catalog) in order to optimize performance. An ADBMS should assist in the initial layout of data on disks and should be able to shift data from one disk to another to even out disk demands. This ability is not present in current DBMSs, however Oracle does provide the ability to move tables while on-line [24].

To make efficient use of system resources, DB2, Oracle and SQL Server permit dynamic *online index reorganization* to reclaim leaf level storage. SQL Server also provides a feature that can automatically shrink the size of allocated database files where more than 25% of the allocated space is not being used [26].

Other self-organizing aspects of SQL Server can be found in the selection of available wizards for data modeling [26]. The Mining Model Wizard is used to create mining models used to analyze data patterns. A Partition Wizard is available to help split data cubes into separate physical partitions on the disk. A Storage Design Wizard can be used to help design aggregations for data cubes while the Usage-Based Optimization Wizard can be used to determine the appropriate aggregations for a given data cube.

2.6 Self-inspecting

Bowing to the principle *if you don't measure it then you don't know it*, an ADBMS should "know itself" in order to make intelligent decisions pertaining to all autonomic features discussed in the previous sections. The DBMS must collect, store and analyze relevant information about its components, performance, and workload. This information should be utilized in optimizing the performance, detecting any potential problems, updating statistics about the stored data, ensuring integrity of data read from disk, scheduling maintenance utilities, and in identifying interesting trends in the workload. The results of this constant inspection should be effectively presented to DBAs (using GUI interface, for example) and be available as input for other autonomic components and operations.

Current DBMSs are rich in self-inspection tools that collect performance information and provide feedback to a DBA. Using the DB2 Health Center or the Oracle Manager Console, a DBA can specify which parts of the system to monitor. Areas include storage usage, memory consumption for caching and sorting, logging behavior, and application concurrency. Performance data are collected and stored in a data warehouse. The monitoring tools examine the system for signs of unhealthiness and issue health alerts via email messages, pages, or records written to the notification log. It can also run corrective scripts or tasks when health alerts are issued. Supporting tools are used to view real-time and historical health alerts, display and enable optional execution of suggested resolution actions, and support configuration changes.

Monitored data are stored and can be used by analysis tools such as The DB2 *Performance Expert*. A collection of reports can be run against the historical data (with correlation of system changes, such as increasing storage) to flag warnings and give advice.

Both DB2 and Oracle provide facilities to collect buffer pool activity data and model changes to the objects in the buffer pools (including sizes) so a DBA can see the effects of changing buffer pool sizes without actually making the changes to the production system.

The *Maintenance Advisor* is a tool that DBAs can use to examine DB2 statistics and make recommendations on what maintenance utilities should be run. It can build scripts or JCL and can schedule maintenance tasks.

Problem determination and diagnosis are supported by utilities such as DB2's *db2support utility* that collects system description about the database configuration, storage devices, network, operating system, and machine specification, and captures a number of database diagnostic files and control structures. All collected data are formatted as HTML pages that are easy to browse by the administrators. DBAs can run db2support also in an interactive mode in which they can describe the problem scenario by answering a sequence of questions driven by a built-in diagnostic decision tree.

Another example of automated inspection is DB2's ability to perform *Sector Consistency Checking* for page I/Os that ensures the integrity of read data by detecting any corruptions caused, for example, by incomplete I/Os. This inspection mechanism exploits consistency bits that verify that the pages read from disk into memory are not "partial pages" or have not been

erroneously modified or corrupted. The net result is a continual validation for the data read by the DBMS.

Oracle, DB2 and SQL Server support utilities to collect statistical information about the stored data (for example, their distributions) to assist the query optimizer develop the most efficient execution plan of a query. In an ADBMS, the collection of these statistics should be dynamic and initiated automatically.

3. Analysis – what is missing?

Despite the many advances that have been made towards autonomic database management systems, much work remains to reduce the amount of human intervention required by these systems. We can summarize the most significant shortcomings in the following points:

- **High need for human input and intelligence**. Current DBMSs provide many tools and utilities to assist the DBA in tasks such as initial configuration, system monitoring and problem analysis, but in most cases these tasks still require a significant amount of input, intelligence and decision making on the part of the DBA. Furthermore, the human inputs required are often error-prone and not permanently reliable given the constant change in the system environment and the characteristics of the workload over time.
- **Need for Dynamic Adaptation**. Tuning advisors, for example, have proven useful in the initial setup of the database system, however the settings do not adapt to changes in the system environment or workload. The tasks of initiating system monitoring and determining that a configuration adjustment is necessary are still, for the most part, left up to the DBA. An ADBMS will constantly collect performance metrics (incurring as little overhead cost as possible) and determine when, and which, resources should be adjusted to maintain or improve performance.
- **Lack of ability to reset DBMS parameters on-line.** Dynamic tuning requires that all resources and configuration parameters be adjustable without system disruption. Although close, DBMSs do not yet provide this capability. Note that being able to reset DBMS parameters dynamically is a mere prerequisite to enable autonomic features but it does not offer any kind of intelligent strategies.
- **Lack of analytical capabilities.** Many of the advisors and tools currently available are based on "rules of thumb" or heuristics that capture the human expertise programmatically. Robust analytical models and accurate prediction mechanisms are required for the more difficult tuning and configuration tasks. Currently corrective action typically depends on the DBA's experience. Future DBMSs should learn from the DBA's experience thus building an extensive knowledge base of information that can be used for problem determination and problem solving.
- **No smart maintenance strategies.** Database utilities such as rebinding, statistics gathering, table and index reorganization and backup are currently provided by the DBMSs. However, an autonomic DBMS must have the ability to predict *when* is the best time to run these utilities (for example, depending on the workload peaks) and how long they will take to finish.
- **Inability to run some operations on-line.** Some of the vital database operations such as deframenting data, updating statistics, and pruning important data structures like indexes can not be performed without bringing the DBMS down.
- **Lack of on-line schema evolution.** This feature should allow changing schema aspects without incurring an outage. Applications are not static. Tables are modified, columns are dropped from tables and indexes, data types for columns are changed, indexes need to be renamed, partitions need to be added, and so on (see [32] for a newly proposed approach). These changes must be made online.
- **Lack of standard interface with other systems.** Current DBMSs do not show adequate enablement of autonomic features that allow smooth integration and synergy between the DBMS, as a middleware, and others such as Web Servers. Standards must be developed for autonomic systems to communicate and share information.
- **Not exploiting the characteristics of the workload.** Most of the current DBMSs overlook analyzing the characteristics of the workload and its behavior over time. ADBMSs should

tune themselves as a function of the present workload intensity, trend, and properties.

- **Trivial security and privacy strategies.** Current security and privacy features do not offer any kind of clever strategies that help the DBMS develop or change its protective plans. For example, an ADBMS should have the ability to analyze and draw intelligent conclusions about the attacks, their types, and their trends. Agrawal et al. propose more interesting ideas to improve DBMSs' privacy **[7]**.

Despite the efforts undertaken by industry-led projects such as IBM's SMART and Microsoft's AutoAdmin, we have not witnessed a real change to the DBMS infrastructure that is necessary for making the transition to a fully autonomic system.

4. Conclusions

Autonomic DBMSs, that is DBMSs that can manage themselves, are an attractive solution to complexity and total cost of ownership problems associated with DBMSs. We examined three popular database products, namely DB2, Oracle and SQL Server, from the viewpoint of autonomic computing systems. We find that, while all three products now contain features of an ADBMS there is still a long way to go before we can claim that DBMSs are autonomic computing systems.

We conclude that ADBMS research should focus in four main areas. The first area is the development of a proper infrastructure to allow the clean introduction of autonomic computing system features. Current research literature proposes two very different paths to ADBMSs. One is a revolutionary approach that argues for a complete redesign of DBMSs with fine-grained components [30][31] or components that provide a RISC-like interface [28]. This RISC-style facilitates individual management of the components and controlled interactions between them. The second approach is evolutionary [27] and identifies a set of phases that existing systems can be taken through in order to become autonomic systems. We feel the evolutionary approach makes the most sense for existing DBMS products.

The second main area of research for ADBMSs is intelligent decision-making tools. It is important that DBMSs become able to independently analyze and act upon the information they collect about their performance. A key component of this progress will be the development of effective mathematical models and feedback control loops that can be used to make more accurate performance prediction and reach better tuning decisions.

The third main area of research is the efficient and automatic collection and exploitation of data about the operation and performance of DBMSs. In order to mange itself an ADBMS will have to increase both the volume of monitoring data and the frequency with which it is collected. An ADBMS will also require workload characterization techniques [3] to automatically extract the necessary information from this data. Statistical models and data mining techniques [1][2] can be useful for exploring interesting properties in the DBMS's workload.

The fourth main area of research is the development of a useful model of the system itself. A model must exist in order for DBMSs to know themselves. The model will have to efficiently represent the resources used by the ADBMS, the relationships between these resources, the workload of the ADBMS and the current state of the ADBMS.

Finally, we do not think that progressing towards ADBMSs will mean the demise of DBAs. It will mean the end of repetitive administrative tasks, freeing DBAs to spend more time on new applications and on the business policies and strategies. Furthermore, DBAs will be needed to evaluate and select recommendations before they are implemented. Once comfortable with system recommendations, DBAs can enable a DBMS to take actions automatically and simply report on them.

5. Acknowledgments

We thank IBM Canada, the Natural Sciences and Engineering Research Council of Canada (NSERC) and Communications and Information Technology Ontario (CITO) for their support.

6. References

[1] Elnaffar, S., Martin, P., and Horman, R. Automatically Classifying Database Workloads. *Proceedings of ACM Conference on Information and Knowledge Management (CIKM ACM '02)*, November 2002.

[2] Elnaffar, S. A Methodology for Auto-Recognizing DBMS Workloads. *Proceedings of Centre for Advanced Studies Conference (CASCON '02)*, October 2002.

[3] Elnaffar, S., and Martin, P. Characterizing Computer Systems' Workloads. *Technical Report 2002-461*, School of Computing, Queen's University, Canada, Dec. 2002.

[4] Office of the Information and Privacy Commissioner, Ontario. Data Mining: Staking a Claim on Your Privacy, January 1998.

[5] S. Castano, M. Fugini, G. Martella, and P. Samarati. *Database Security*. AddisonWesley, 1995.

[6] C. Landwehr. "Formal Models of Computer Security". *ACM Computing Surveys*, 13(3):247–278, 1981.

[7] Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y. "Hippocratic Databases". *VLDB 2002*, Hong Kong, China. VLDB Endowment, in press 2002.

[8] Rao, J., Zhang, C., Lohman, G., Megiddo, N. , "Automating Physical Database Design in a Parallel Database System", *Proc. 2002 ACM SIGMOD*, Madison, WI, 2002.

[9] Gassner, P., Lohman, G.M., Schiefer, K.B. and, Wang, Y. "Query Optimization in the IBM DB2 Family", *IEEE Data Engineering Bulletin*, 16(4), 1993, pp. 4-18.

[10] M. Stillger, G. M. Lohman, V. Markl, M. and, Kandil, "LEO - DB2's LEarning Optimizer", *VLDB 2001*, Rome, Italy, pp. 19-28.

[11] E. Kwan, S. Lightstone, A Storm and, L. Wu, IBM Server Group, "Automatic Configuration for IBM DB2 Universal Database", online, http://www.redbooks.ibm.com/redpapers/pdfs/redp0441.pdf.

[12] G. Lohman, G. Valentin, D. Zilio, M. Zuliani and, A. Skelly, "DB2 Advisor: An optimizer Smart Enough to Recommend Its Own Indexes", *Proceedings, 16th IEEE Conference on Data Engineering*, San Diego, CA, 2000.

[13] B. Schiefer and G. Valentin. "DB2 Universal Database Performance Tuning", *IEEE Data Engineering Bulletin*, 22(2), June 1999, pp. 12-19.

[14] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. "Automated Selection of Materialized Views and Indexes for SQL Databases", *VLDB 2000*, pp. 496-505.

[15] Guy M. Lohman, Sam Lightstone, "SMART: Making DB2 (More) Autonomic". *VLDB 2002*, pp. 877-879.

[16] H. Pirahesh, J. M. Hellerstein, W. Hasan, "Extensible/Rule Based Query Rewrite Optimization in Starburst", *Procs. 1992 ACM SIGMOD Conference*, 1992, pp. 39-48.

[17] H. Pirahesh, T. Y. C. Leung, W. Hasan, "A Rule Engine for Query Transformation in Starburst and IBM DB2 C/S DBMS", *Procs. 1997 IEEE Intl. Conf. On Data Engineering*, 1997, pp. 391-400.

[18] P. Gassner, G. M. Lohman, K. B. Schiefer, Y. Wang. "Query Optimization in the IBM DB2 Family", *IEEE Data Engineering Bulletin*, 16(4), 1993, pp. 4-18.

[19] "Database Cost of Ownership Study," The Aberdeen Group 1998. http://relay.bvk.co.yu/progress/aberdeen/aberdeen.htm.

[20] D.H. Brown Associates, "DB2 UDB vs. Oracle8i: Total Cost of Ownership," D.H. Brown Associates, Inc., Port Chester, NY., December 2000. http://www.breakthroughdb2.com/.

[21] Ashraf Aboulnaga and Surajit Chaudhuri. "Self-tuning Histograms: Building Histograms Without Looking at Data", *Proceedings of ACM SIGMOD*, Philadelphia, 1999.

[22] Surajit Chaudhuri and Vivek Narasayya. "Automating Statistics Management for Query Optimizers". *Proceedings of 16th International Conference on Data Engineering*, San Diego, USA 2000.

[23] Query Optimization in Oracle 9i. An Oracle White Paper, February 2002. http://technet.oracle.com/products/bi/pdf/o9i_optimization_twp.pdf.

[24] Oracle 9i Manageability Features. An Oracle White Paper, September 2001. http://www.oracle.com/ip/deploy/database/oracle9i/collateral/ma_bwp10.pdf

[25] Oracle 9i Materialized Views. An Oracle White Paper, May 2001. http://technet.oracle.com/products/oracle9i/pdf/o9i_mv.pdf

[26] Microsoft SQL Server 2000 Documentation, Microsoft Corporation, 2002.

[27] A. G. Ganek and T. A. Corbi. "The Dawning of the Autonomic Computing Era". *IBM Systems Journal*, 42, 1, March 2003.

[28] Gerhard Weikum, Axel Mönkeberg, Christof Hasse, Peter Zabback: "Self-tuning Database Technology and Information Services: From Wishful Thinking to Viable Engineering". *VLDB 2002*, pp. 20-31.

[29] IBM, DB2 Universal Database Version 8.1 Administration Guide: Performance, IBM Corporation, 2003.

[30] J. McCann, "The Database Machine: Old Story, New Slant?" *Proceedings of the first Biennial Conference on Innovative Data Systems Research*, VLDB, January 5-8 2003.

[31] S. Harizopoulos and A. Ailamaki, "A Case for Staged Database Systems" *Proceedings of the first Biennial Conference on Innovative Data Systems Research*, VLDB, January 5-8 2003.

[32] P. Bernstein, "Applying Model Management to Classical Meta Data Problems" *Proceedings of the first Biennial Conference on Innovative Data Systems Research*, VLDB, January 5-8 2003.