Autonomic Database Management: State of the Art and Future Trends

Katarina Grolinger, Miriam A.M. Capretz
Department of Electrical and Computer Engineering, Faculty of Engineering
The University of Western Ontario
London, ON, Canada N6A 5B9
{kgroling, mcapretz}@uwo.ca

Abstract

In recent years, Database Management Systems (DBMS) have increased significantly in size and complexity, increasing the extent to which database administration is a time-consuming and expensive task. Database Administrator (DBA) expenses have become a significant part of the total cost of ownership. This results in the need to develop Autonomous Database Management systems (ADBMS) that would manage themselves without human intervention. Accordingly, this paper evaluates the current state of autonomous database systems and identifies gaps and challenges in the achievement of fully autonomic databases. In addition to highlighting technical challenges and gaps, we identify one human factor, gaining the trust of DBAs, as a major obstacle. Without human acceptance and trust, the goal of achieving fully autonomic databases cannot be realized.

1 INTRODUCTION

The size and complexity of databases have been increasing significantly in recent years. Innovations in modern hardware and software have enabled systems with hundreds of disks and numerous CPUs, thus allowing databases to grow to previously unimaginable sizes, such as the 8 exabytes in Oracle 11g [1]. In fact, the increase of Internet use for activities such as online banking, trading and shopping has increased the number of concurrent users and caused the terabyte-sized database to become common.

With each new database version, vendors are releasing new features and data structures as well as table and index types. These new features provide databases with great strength and flexibility; however, they also cause challenges in database management. The required Database Administrator (DBA) skill set is growing, and specialized database administrators, such as security or warehouse administrators, are becoming common. Nevertheless, database performance depends extensively on the individual DBA's skills.

Consequently, a major part of today's database expenses relate to DBAs. The solution for reducing this cost involves autonomic computing systems that manage themselves. "Autonomic systems are computer systems that can regulate themselves much in the same way as our autonomic nervous system regulates and protects our bodies" [2]. Accordingly, Autonomic Database Management Systems (ADBMS) should be able to self-regulate, including the ability to self-configure and self-optimize as well as self-protect and self-heal without human intervention.

In recent years, there have been extensive research efforts in the area of ADBMS, and commercial database systems, especially Oracle 10g, IBM DB2 and Microsoft SQL Server, have made significant steps towards ADBMS. However, most efforts focus on specific autonomic features rather than ADBMS as a whole. In particular, significant advances have occurred in areas where DBA work was repetitive and/or time consuming, including memory management and index recommendation. However, the long-term goal of attaining fully autonomous DBMSs must be achieved through many small steps.

This paper evaluates the current state of database management system autonomy in leading commercial databases. Specifically, the major challenges in achieving autonomic databases and gaps in the current research are identified. Unlike the objective of Mateen et al. [3], who compare DB2, Oracle and SQL server and assign maturity values to a variety of autonomic features, the goal of our work is to observe the current state of autonomy and identify areas where improvements are necessary.

The paper is organized in the following manner; first, we present our view of autonomic categories in Section 2 and examine the current state of autonomy in Section 3. Section 4 identifies challenges and technical gaps, while Section 5 discusses human factors in ADBMS. Finally, conclusions and future trends are presented in Section 6.

2 ADBMS REQUIREMENTS CATEGORIES

Typically, research related to ADBMS focuses on a specific self-management feature [4][5]. However, some works consider autonomous systems as a whole or a significant subset of autonomous features [6][7]. In particular, substantial advances have been made in the area of memory optimization [1], query tuning [8] and dynamic tuning for workloads [4] [5].

Requirements of the entire autonomous database system are commonly grouped into four self-CHOP categories [9]: self-configuring, self-healing, self-optimizing and self-protecting. Furthermore, the areas of self-organizing and self-inspecting are occasionally treated as individual categories [10], while some authors consider these categories as subcategories of the four self-CHOP categories. However, the boundaries between categories are very vague and various authors put the same feature into different categories. For example, while configuration parameters belong to the category of self-configuring, they are also considered as self-optimizing,

since an optimal value must be selected for a good database performance.

Self-knowledge, often referred to as self-inspection or reporting, is not part of the four self-CHOP categories; however, we consider this factor as the foundation for all four categories. Therefore, in order to include self-knowledge, we have modified the self-CHOP categories to self-KCHOP; this revised depiction of ADBMS is illustrated in Figure 1. In this figure, self-knowledge is the centre of the system and the foundation for all other categories. Before self-configuring or self-optimizing can occur, the database must have self-knowledge, which includes information about its current setup, its current and usual workload and its available resources as well as any other factor that could influence its performance. For example health monitors infer about the database state by relying on information provided by self-inspection. The goal-driven self-management system of Holze and Ritter [7] uses the system model as the knowledge base for the autonomic solution.

3 AUTONOMIC FEATURES OF CURRENT DBMS

The autonomic features of current database systems are reviewed in the five self-KCHOP categories. The features included in each category are not inclusive sets for the category; rather the features comprise the representative set that enables us to assess the overall state of autonomy as well as to identify challenges and research gaps in database management autonomy. The examples provided from specific database vendors are solely for illustration purposes and do not imply that other vendors do not have similar features.

3.1 Self-knowledge

Self-knowledge is the knowledge that the database possesses about itself. It is a prerequisite for autonomic database management, as the database needs to understand itself in order to make decisions about any of the four self-CHOP categories.

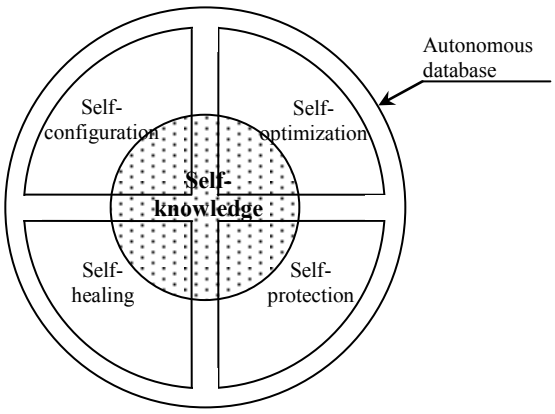

Fig. 1: ADBMS: The five self-KCHOP

Self-knowledge is achieved through the use of monitors that track necessary information for automatic and manual database management, such as events, database activities, resource usage and running processes.

For this purpose Oracle uses an Automatic Workload Repository (AWR), which captures and stores snapshots of performance statistics at a preset time intervals. The Automatic Database Diagnostic Monitor (ADDM) analyzes data in the AWR and reports the findings for the observed time period. In addition to reporting, the ADDM provides warnings and recommendations that are essential for improving database performance. However, despite the work of the ADDM, the DBA is still responsible for the final decision of whether or not to implement the recommended changes. A similar inspection and notification process is used with DB2 Health Advisor.

3.2 Self-configuring

Self-configuring is the ability of the system to configure itself in order to achieve its goal. In this process, the system must detect changes in the environment and adapt the configuration accordingly. Since the adaptation needs to achieve optimal performance, the category of self-configuring is inseparable from self-optimization. Some of the self-configuring features include the following:

- Memory management. In the past, memory management required extensive human involvement and substantial time investments. Today, most systems have dynamic self-tuning memory management that does not require human intervention. However, the DBA can override automatic memory management and manually control memory.
- Dynamic configuration parameters. All database vendors have made the effort to make most of the configuring parameters dynamic, thus indicating that the database does not require restarting in order for the parameter to take effect.
- Supporting objects, such as indexes, materialized views, clusters and partitions are the foundation for a good performance. Commercial database systems provide recommendations through advisors. Nearly all of these systems include indexes, materialized views and SQL profiles, while only some of them, such as DB2 Design Advisor or Oracle Partition Advisor, include advice for clustering and partitioning. Oracle 11g includes Automatic SQL Tuning, which, in addition to making recommendations, also implements changes [8]. However, this ability is limited to the SQL Profile.
- Self-organizing entails the ability of the database to restructure its objects for maintaining optimal performance. Specifically, this feature deals with fragmentation problems. In the past, self-organization was considered as a separate category [10], since it required significant human effort. However, most

current database systems have managed to decrease fragmentation issues by modifying the way in which they handle space allocation and de-allocation.

- Storage advisors reduce DBA involvement in the process of specifying storage for database objects. For instance, a highly autonomic storage advisor is Oracle Automatic Storage Management (ASM). The DBA only specifies the disks available for the storage, while ASM decides how to split database objects between disks and determines which objects are stored in which places, thus ensuring optimal storage performance. Storage management is sometimes considered part of self-organization, as it deals with issues concerning space.

3.3 Self-optimizing

Self-optimization involves the ability of the database to configure and adapt itself in order to attain optimal performance for the current environment, workload and available resources. This ability is the performance tuning category, and it includes the following features:

- The Query Optimizer determines the optimal query execution plan. Although this characteristic was the first truly autonomic database feature, there are still significant research efforts in this area [11], as an efficient query plan is essential for a strong database performance.
- Statistics management. The Query Optimizer requires up to date statistics to make decisions about optimal query execution plans. Statistics collections in most DBMS are fully automated.
- Resource-intensive process control. Processes such as data loads, backups and batch processes require significant resources, which may decrease the speed of production processes. DB2 manages its process control through Utility Throttling, which occurs when the DBA specifies the extent to which the utility can impact database performance. Oracle uses Resource Plans, where the DBA assigns priorities for different consumer groups.
- Workload management systems, sometimes referred to as query management, control the flow of queries into the database. These systems strive for a more efficient allocation of system resources by assigning different priorities to queries based on business requirements, limiting resources for query categories and tracking runaway queries [4]. The DB2 Query Patroller (DB2 QP) is a DB2 workload manager, the SQL server uses Query Governor and Oracle has Resource Manager.
- Task scheduling is performed in Oracle and DB2 through windows that represent time periods. Specifically, the DBA assigns a task for the time window, and the task is executed when that window is open. For instance, in most systems, the highest load occurs during the day, so the window of time occurs during the night and the tasks assigned to it are executed. Today, with an increasing emphasis on a twenty-four hour society, the maintenance windows are disappearing, so it is becoming more difficult to perform maintenance or batch operations.

3.4 Self-Healing

Self-healing involves the database's ability to recover itself after failures. In particular, the database needs to acknowledge the occurrence of the failure, determine whether a full or incremental recovery is needed, recognize the resources that are available for the recovery and perform the recovery. The self-healing features include:

- Automatic Restore. The DBMS searches through backup files and restores the database as needed. This feature is initiated and usually supervised by DBA.
- Recovery of selected database objects. The database's ability to recover only certain objects shortens the recovery process increasing availability. For example, Oracle 11g can recover on the table level or even on the database block level.
- Recovery Advisors/Experts assist DBAs in recovering the database in easier and faster ways.
- Although standby databases are not a true self-healing feature, they enable the DBMS to continue running during failure. Specifically, a standby database is a copy of the main database, which enables the user to utilize this substitute database in the event of failure.
- The grid environment alleviates recovery concerns. In the Oracle Real Application Clusters (RAC) environment, a single database runs on several database servers, enabling a single server to be taken offline while the database remains functional.
- Health Monitoring Utilities monitor the database activity, and if an unwanted situation occurs or is close to occurring, the utility triggers an alarm. These utilities are known by different names: in DB2, the utility is called a Health Monitor, and in Oracle, it is an Automatic Database Diagnostic Monitor (ADDM). In addition to triggering alerts in a variety of forms, including email or pager, monitors can provide recommendations for improvements.

3.5 Self-protecting

A Self-protecting database can protect itself from unwanted activity and includes:

- Authentication mechanisms that prevent unauthorized access to the database.
- Privilege control. Each user can only access the parts of the database necessary for performing certain tasks. Specifically, the level of control can be at the table or row level. The Oracle 11g Virtual Private Database feature uses policies for fine-grained access control.
- Encryption. The database manages the encryption and decryption of data. Existing databases or their parts can be encrypted without the necessity of changing the

application. Existing applications still see decrypted data, as the data is transparently decrypted by the database.

4 CHALLENGES AND GAPS IN AUTONOMIC DATABASE SYSTEMS

The high level of complexity in database systems and the numerous features of databases complicate the task of achieving database autonomy. With each new database version, every vendor releases new features that are often intended for specific situations. For example, Oracle 11g offers special functionalities for medical imaging and geospatial data. These highly specific and less frequently used features are not significant for database autonomy and efforts towards autonomy focus on the features that are commonly used and/or require extensive DBA time. Thus, some highly specialized features will remain manual, as the efforts and costs of automating these features may outweigh the benefits of autonomy.

In the following subsections, we discuss the ADBMS features, focusing on their limitations, advantages, disadvantages and areas for future improvement.

4.1 Challenges of Self-knowledge

The quality of self-optimization and self-configuration depends on the quantity of the collected information. More detailed monitoring enables better optimization, but it also requires more resources. The way in which the monitoring can be performed is limited by the need to monitor without an impact on database performance. Specifically, the monitoring must be done within the database engine or it will utilize significant resources.

Generic approaches that are designed to be used with a variety of database systems [6] and implemented as independent entities require the monitoring to be performed outside of the DBMS engine. Since these approaches require significant resources, such approaches have not been accepted in practice. The SQLCM approach [12] provides a framework for monitoring from within database engines, thereby decreasing the need for resources. However, although the SQLCM framework is generic, it requires a separate implementation for each database engine.

Database vendors have implemented monitoring within database engines where there is fast and easy access to monitored information. In this case, information in the memory is accessed directly and the beginning and end of events can be trapped from within the database code. Nevertheless, if information pooling, writing and analyzing is performed frequently and extensive data is collected and analyzed, a significant database load can be incurred. To alleviate this load, vendors provide different monitoring levels: a typical monitoring level, which is used in production, and an extensive monitoring level, which is intended for database tuning. However, since tuning often needs to be performed on the production system during regular working hours, extensive monitoring may not be possible.

SQL Anywhere utilized an approach where a database designed for embedded systems was installed along with the application [13]. An embedded database requires little or no human maintenance; these databases are typically smaller databases, and, as such, they do not need the flexibility and advanced functionalities of larger databases. In addition, embedded databases usually do not have a high number of users, which further simplifies the administration processes. By sacrificing functionality and flexibility, SQL Anywhere is able to achieve high autonomy. However, due to its limitations, this approach is only useful for a limited set of applications.

4.2 Challenges of Self-configuring

A self-configuring feature that requires little DBA involvement is memory management. In the past, memory management was an area where DBAs spent significant time in properly configuring the database, and once memory was configured, it required DBA involvement when workload changes occurred. Today, most systems do not require DBA involvement in memory tuning. However, there are some exceptions to this rule, and the DBA is still able to manually tune memory if needed. Database vendors continue to work on improving memory management algorithms.

Significant advances have also been made in regards to supporting database objects, especially indexes and materialized views. All major database systems have advisors that provide recommendations for indexes, materialized views and SQL profiles. However, there are not as many advances for the support of partitioning and clustering. Because of the wide range of different partitioning methods, such as range, hash, list, range-hash and range-list, as well as clustering types, such as index, hash, and sorted hash, it is difficult to achieve autonomy. At the same time, with the growth of database sizes, clustering and partition have become common and DBAs spend significant time in determining the appropriate structures.

Automatic Storage Management (ASM), which completely assumes the burden of storage management, is available for most major database systems. However, ASMs contain drawbacks; for example, Oracle ASM uses a separate ASM database to manage storage information. Although ASM requires minimal management, the presence of an additional database requiring management presents another burden on the DBA. Furthermore, ASM database is another point of failure: if the ASM database fails the production DB will fail as well. Also, when the standby database is used, ASM requires the standby database as well. These drawbacks of ASM have resulted is its limited application in practice.

4.3 Challenges of Self-optimization

The first truly autonomic feature is the Query Optimizer, whose quality has a significant impact on the database performance. Thus, although the query optimizer has been in existence for a long time and it is fully autonomic, efforts on its improvements continue. Specifically, the releases of new database structures and data types require adjustments to the query optimizer.

The control of resource-intensive processes is performed using a variety of approaches. The selection of a simple approach increases the level of autonomy but decreases the efficiency and flexibility. Thus, efficient approaches currently require significant DBA involvement. Consequently, current efforts aim to simplify the control of resource intensive processes while maintaining flexibility and efficiency.

The scheduling of administrative tasks has become challenging with the 24/7 systems, which have continually high workloads with few low-workload periods during which maintenance operations would typically occur. As a result, maintenance operations are treated similarly to other resource-intensive operations, where their resource usage is controlled using throttling or a similar utility. Additionally, these 24/7 systems are expected to result in the increased use of auxiliary, standby databases during the maintenance of the main database.

4.4 Challenges of Self-healing

The only area of self-healing that received significant attention is the aspect of health monitoring and notification. In this area, database recovery is not considered a major burden on database administrators because it is not a common, repetitive task. The use of standby databases and grid technology enables a system to continue running in case of database failures, thus relieving the pressure on the DBA to recover the database quickly.

Health monitors typically trigger alerts when unwanted situations arise. The challenge in this process occurs in deciding when the alert is needed. Some alerts rely on thresholds, where the alert is triggered when a certain threshold is reached. The drawback of the threshold approach involves the DBA's extensive work in establishing thresholds. While default thresholds exist, they are only applicable in limited situations. In addition, other alerts rely on the comparison of the current database state with the baseline, which is established by the DBA. However, baselines do not require as much DBA involvement as do threshold alerts. During satisfactory database operation, the DBA tells the database that this state is their required operation level, or, the baseline. This approach contains the drawback of requiring the database to be in a satisfactory performance state for the baseline to be set. While this approach works in situations when the database is running as required and DBA is trying to keep it in such a state, it is problematic when the database is in its initial stages and the normal operation level is not known.

4.5 Challenges of Self-protecting

The feature of self-protecting concentrates on preventing access to unauthorized users. Once a user has access to the database, there is a little preventing him/her from doing major harm. The attacks from within the database can be malicious, which occur when the user intentionally tries to harm the database, or non-malicious, which happens when the user unintentionally harms the database due to user error. Users, including DBAs and developers, can make a mistake that can compromise data or monopolize resources. Specifically, they can initiate a process that slows down the database or they can change thousands of records by error. Attacks from within the database, whether intentional or unintentional, are difficult to recognize, as it is challenging to distinguish if a particular query is an attack or is merely a resource-intensive operation. Intrusion detection research attempts to identify actions that can compromise the confidentiality, integrity or availability of a resource.

While Oracle flashback technology can help the database to recover from some errors, it cannot prevent such errors from happening. DBAs try to reduce the harm that users can inflict upon the database by limiting access and resources; however, this process is intensive with regards to human involvement, and quite often, it does not even truly protect the database. In fact, the effort to limit access and resources can even cause problems, especially when the user's work requires more resources than the DBA anticipated.

5 HUMAN FACTOR IN ADBMS

While the human factor is an essential part of advancing toward autonomic databases, it is frequently not considered. Current commercial databases provide powerful tools to help DBAs in the administration process and a variety of autonomic or semi-autonomic features.

However, DBAs frequently choose not to use these features, especially in the case of legacy systems. When legacy databases were first developed, most autonomic features did not exist. Subsequent upgrades to new database versions attempted to preserve as much of the current setup as possible with the belief that extensive changes may degrade performance. Even with newer systems, DBAs are extremely cautious with their decisions to use the latest autonomic features. For instance, rather than implementing a feature in its first release, they might wait for subsequent releases, expecting any potential issues to be resolved by that time. DBAs are more accepting of self-knowledge features, as they represent a smaller risk than self-configuring or self-

optimizing features, which change the database environment and potentially cause significant damage to the database.

This resentment toward new features is much stronger in the database field than it is in software development. DBAs feel responsible for the database, and they consider stability and reliability more important than the availability of new features or the potential for performance improvement. Sometimes, DBAs prefer to use older established methods rather than the newer and more efficient ones, if they do not completely trust the new methods.

Therefore, one of the main challenges in the process toward autonomous databases involves gaining the trust of DBAs. In addition to providing autonomic features, efforts need to be made for facilitating the acceptance of those new features. Belknap et al. [8] consider reporting and GUI support even more essential for autonomic features than for manual ones. Specifically, these authors believe that a lack of clarity in the features' presentation will create a lack of trust and the feature will be disabled. Hence, presentation clarity is a step towards gaining the trust of DBAs; however, additional research must be conducted in order to identify other ways for building trust. Autonomic features may be accompanied with a variety of built-in safeguards, and consequently, efforts should ensure that DBAs are educated about the safeguards as well as the potential benefits of the feature.

One way of gaining the DBAs' trust involves providing the ability to review scripts semi-autonomic features provide before their final execution. When database vendors release powerful GUI tools for database management, DBAs are initially unclear about how the database responds to their GUI clicks. Therefore, database vendors provide DBAs with an opportunity to review the script GUI creates and, if needed, to manually modify the script before its execution.

6 CONCLUSIONS AND FUTURE TRENDS

The size and complexity of database systems have being growing significantly during the last decade, and it is expected that these factors will be increasing even more rapidly in the near future. Database administration is becoming more complex and time-consuming, creating the necessity for autonomous database systems. Even though significant advances have been made towards autonomous databases, further improvements are needed, especially in the following areas:

- New approaches are necessary for facilitating the development of trust in using autonomic feature in their early releases. Without this step, the progress towards achieving highly autonomic databases will be slowed down.
- Since advisors recommend a variety of solutions, future efforts need to implement these solutions.
- Although a variety of advisors and monitors exist, their integration is limited. Thus, further integration is necessary for the provision of comprehensive autonomic solutions.
- Advances in monitoring are necessary for enabling extensive data collection without an impact on performance.

The increasing trend towards database autonomy will not make DBAs obsolete. Rather, it will save time for DBAs, thus enabling them to handle different tasks, resulting in better, faster and more reliable systems.

7 REFERENCES

[1] Oracle Database 11g (11.2) Documentation, http://www.oracle.com/technetwork/database/enterprise-edition/documentation/index.html, 2009.

[2] G. Rabinovitch, D. Wiese, "Non-Linear Optimization of Performance Functions for Autonomic Database Performance Tuning", *Proceedings of the Third International Conference on Autonomic and Autonomous Systems*, pp. 48-54, 2007.

[3] A. Mateen, B. Raza, M. Sher, M.M. Awais, T. Hussain, "Evolution of Autonomic Database Management Systems", *In the Proceedings of the 2nd International Conference on Computer and Automation Engineering*, pp. 33-37, 2010.

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

[5] S. Narayanan, F. Waas, "Dynamic Prioritization of Database Queries", *Proceedings of 27th International Conference on Data Engineering*, pp. 1232-1241, 2011.

[6] S. Ramanujam, M.A.M. Capretz, "Design of a Multi-Agent System for Autonomous Database Administration", *Proceedings of Canadian Conference on Electrical and Computer Engineering*, Volume 2, pp. 1167-1170, 2004.

[7] M. Holze, N. Rittera. , System Models for Goal-Driven Self-Management in Autonomic Databases, *Data & Knowledge Engineering*, Volume 70, Issue 8, pp. 685-701, 2011.

[8] P. Belknap, B. Dageville, K. Dias, K. Yagoub, "Self-Tuning for SQL Performance in Oracle Database 11g", *Proceeding of IEEE 25th International Conference on Data Engineering*, pp. 1694-1700, 2009.

[9] C.M. Garcia-Arellano, S.S. Lightstone, G.M. Lohman, V. Markl, A.J. Storm. , Autonomic Features of the IBM DB2 Universal Database for Linux, UNIX, and Windows, *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Volume 36, Issue 3, pp. 365-376, 2006.

[10] S. Elnaffar, W. Powley, D. Benoit, P. Martin, "Today's DBMSs: How Autonomic are they", *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, pp. 651-655, 2003.

[11] B. Raza, A. Mateen, M. Sher, M.M. Awais, T. Hussain, "Autonomic View of Query Optimizers in Database Management Systems", *Proceedings of Eighth ACIS International Conference on Software Engineering Research, Management and Applications*, pp. 3-8, 2010.

[12] S. Chaudhuri, A.C. Konig, V. Narasayya, "SQLCM: A Continuous Monitoring Framework for Relational Database Engines", *Proceedings of the 20th International Conference on Data Engineering*, pp. 473-484, 2004.

[13] I.T. Bowman, P. Bumbulis, D. Farrar, A.K. Goel, B. Lucier, A. Nica, et al., "SQL Anywhere: A Holistic Approach to Database Self-Management", *Proceeding of IEEE 23rd International Conference on Data Engineering Workshop*, pp. 414-423, 2007.