Usability and design considerations for an autonomic relational database management system

by R. Telford
R. Horman
S. Lightstone
N. Markov
S. O'Connell
G. Lohman

Autonomic systems offer numerous advantages over non-autonomic systems, and many of these advantages relate to ease of use. The advantages regarding ease of use include reducing the number of low-level system administration tasks, simplifying the system administrator's interface, handling exceptions which would otherwise have resulted in system alerts, and the learning, by the system, of actions taken by the administrator. However, human intervention must still be factored in, and care must be taken in the design of autonomic systems not to make the system administrator's task more difficult. This paper examines the ease-of-use ramifications of autonomic computing in the context of relational databases in general, and of the IBM® DB2® Universal Database™ Version 8.1 autonomic computing system in particular.

In October of 2002, IBM made two key announcements. The first was a vision of "e-business on demand," a blueprint for the future of computing. e-business on demand computing is characterized by four key traits—integration of systems, openness, virtualization of hardware and software resources, and autonomic computing.[1] The second announcement concerned autonomic computing itself—IBM announced an organization, and a cross-company initiative, to deliver autonomic computing systems.[2] How does autonomic computing relate to ease of use? This paper answers that question by describing a real-life example of an autonomic system with strong ease-of-use characteristics. This system, the IBM DB2* Universal Database* (UDB) Version 8.1 autonomic computing system, employs a number of usability features combined with autonomic technologies to deliver an administrative interface unlike anything else in the industry.

Autonomic computing and ease of use

Autonomic computing is all about self-managing systems. At the core of the autonomic computing initiative is the concept that computers need to be more self-configuring, self-healing, self-optimizing, and self-protecting in order to reduce the overall complexity of a system. Autonomic computing is described as a "closed loop" system, which includes a "monitor-analyze-plan-and-execute" process in order to make decisions.

From an ease-of-use perspective, autonomic computing offers a leap forward in the following ways:

Self-configuration. Rarely does an IT component work "out of the box." As part of the installation process, there is always a certain amount of configuration required by the user in order to allow the component to work appropriately within the environment in which it is installed. In an autonomic computing system, the system itself can configure its components. Maintaining the current state of a system's configuration (and that of each component) at all times,

0018-8670/03/$5.00 © 2003 IBM

the system has the ability to predict the impact of a change to the configuration and learn from the results of such changes. It learns the ramifications of each change and decides when it is appropriate to make them.

Self-optimization. Today, much of an administrator's time is spent keeping a system "tuned" to changing workloads. What works today for one type of workload may no longer be optimal if the workload grows, shrinks, or changes. Manual optimization of systems is both time-consuming and difficult. Quite often it is based on human trial-and-error rather than careful analysis and planning. Using the principle of policy-based decision-making, an autonomic system tunes itself to meet a set of objectives defined in the policies. Policies direct autonomic systems in finding computational partners, prioritizing limited resources, maintaining security, and recovering from failures. Policy specification, though not a new field, takes on an entirely new significance within the context of autonomic computing.

Self-protection. The state of the art today in many system protection infrastructures involves the use of an alert mechanism to prompt an administrator's response when an intrusion detection system detects what appears to be a potential threat. By reducing or eliminating the role of the administrator in responding to these alerts and taking appropriate security measures automatically, autonomic systems can reduce the time required to re-establish system security.

Self-healing. In complex e-business infrastructures, it is critical for the system to have the ability to detect inter-component errors and perform "root cause analysis" to help correct them. Because of the heterogeneous nature of an e-business infrastructure spanning multiple components, there is no standard way to correlate and trace transactions across the system. Each component logs information in its own format, requiring the labor-intensive task of manually correlating events across the infrastructure. In an autonomic system, this information is logged and correlated at the system level, allowing the system to perform event correlation and root cause analysis.

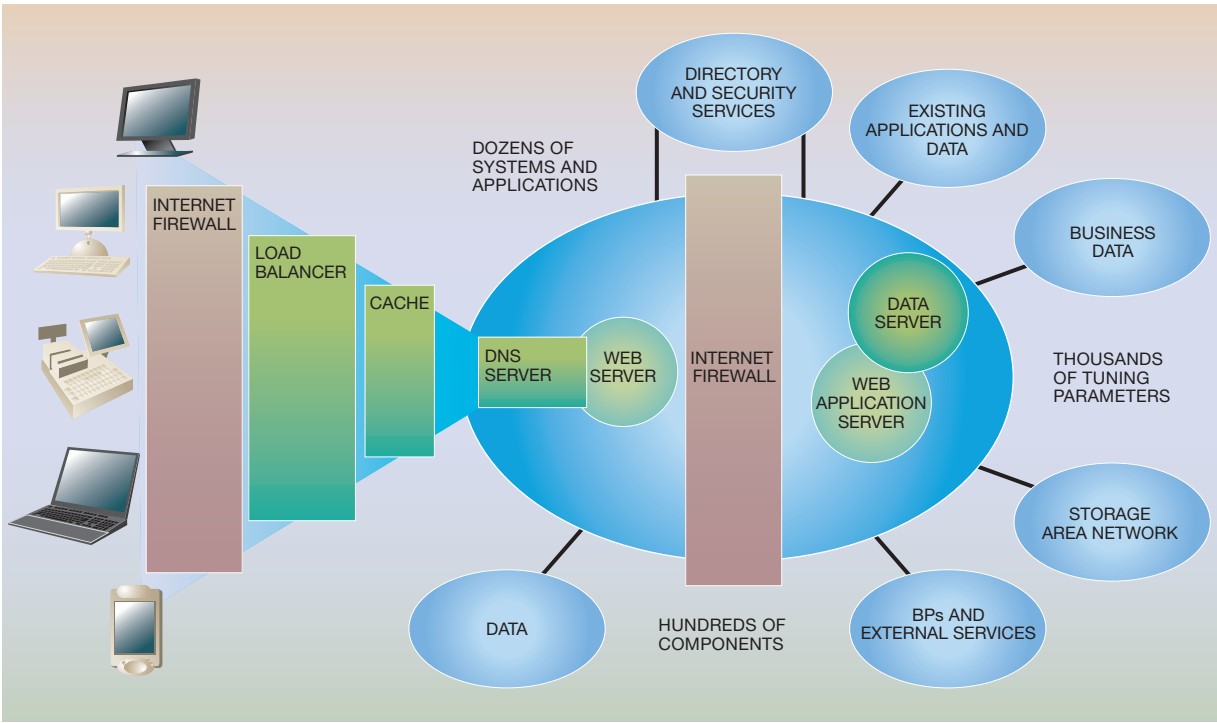
One of the key ease-of-use values of an autonomic system derives from the ability to lower the burden of mundane tasks and allow the administrator to focus on higher-order problems and responsibilities. Another key ease-of-use attribute of autonomic systems is the simplification of the administrator's interface. Autonomic computing allows for a less complex and less cluttered view of the system infrastructure because the display of much of what is currently presented to an administrator is no longer necessary. Examples of such simplifications include:

1. Allowing the system to self-query for information rather than prompting the administrator. Through the development of "autonomic widgets" for system consoles, autonomic systems will be able to query themselves for answers to questions that today must be answered by the administrator. Information about the system configuration, network settings (such as IP addresses), and system settings can all be gathered autonomically and not require human intervention.
2. Raising the threshold for alerts delivered to the administrator's console. Autonomic computing allows for advanced filtering of events and alerts that occur in the system. Through pattern recognition and similar technologies, an autonomic system can filter out "false positives" before they reach the console and handle many alerts automatically.
3. Allowing the system to act using learned behaviors. An autonomic system monitors the actions taken by administrators and "learns" from them; that is, when a certain event occurs and the administrator responds, this event/response association is learned. The next time such an event occurs, the system can make recommendations regarding what action to take in response. Over time, the system will be able to take many actions without needing human intervention.

Autonomic computing has potential problem areas with respect to ease of use, and systems need to be designed with care. Adding autonomic capabilities to a system can actually make the user experience more challenging—exactly the opposite of what was intended. As stated by Russell et al., ". . . autonomic computing makes effective design of the user experience even more challenging and critical than it is now. The reason is that autonomic actions taken by the system must be understandable by the user and capable of review, revision, and alteration. Because such actions are often made autonomously, a heavy burden is placed on the ability of the system to explain what it is doing and why."[3]

As enterprises adopt and integrate more technology components and realize greater production efficien-

Figure 1 Complex heterogeneous systems



cies, their IT infrastructures grow in complexity. Quite simply, there is a growing amount of hardware and software to install and configure, many more elements that need to work together properly, and a great deal of pressure to keep it all up and running on a daily basis. It is interesting to note that in such complex, heterogeneous systems (as shown in Figure 1), the total cost of ownership (TCO) is increasingly dominated by human costs.

Through autonomic computing, system management is simplified not only by improved ease of use, but also by a marked reduction in problems caused through human error, and the subsequent reduction in human interaction required for corrective action. A brief examination of the causes of system crashes (though certainly only a subset of possible system errors) is illustrative. Recent projections suggest that the percentage of system failures caused by human error has grown since 1985 from 15 percent to an estimated 69 percent in 2001, as shown in Figure 2. Through autonomic computing, many of the configuration and maintenance operations that human administrators previously had to perform can be automated, resulting in fewer system failures (because the autonomic system will neither forget nor make random errors), and, therefore, a reduction in outages. Of course, autonomic systems can make mistakes as well. The advantage of autonomic computing is, in part, the notion of consistent control—that is, that the autonomic system never forgets and never suffers from random faults, as human administrators certainly do. Patterson et al.[4] suggest that the sharp increase in human-related causes for system crashes is due largely to the increased complexity of heterogeneous systems and middleware. Autonomic computing offers a paradigm for computer systems to deal with this complexity in ways that human beings simply cannot.

A number of initiatives within IBM and the IT industry[5] are currently focused on using autonomic computing as a way to address system complexity and overall ease of use. The experience of IBM's database technology team offers an excellent example of how a system can be transformed and ease of use can be improved through autonomic computing.

Figure 2 Causes of system crashes

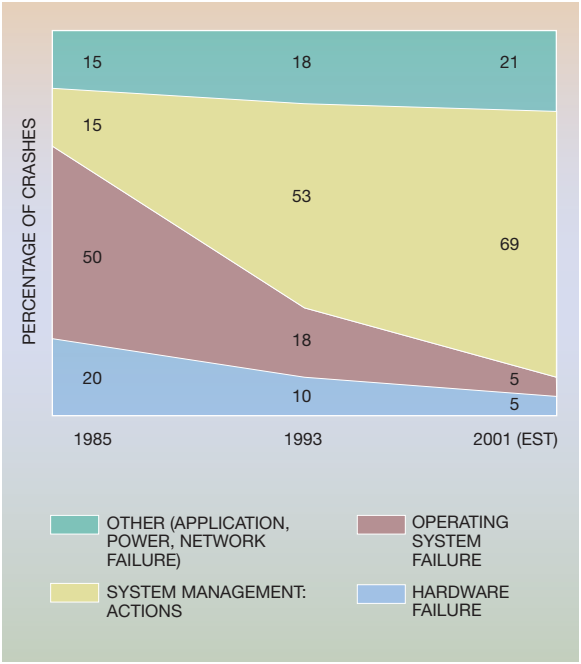


Figure 3 Results of D. H. Brown TCO study for data warehousing

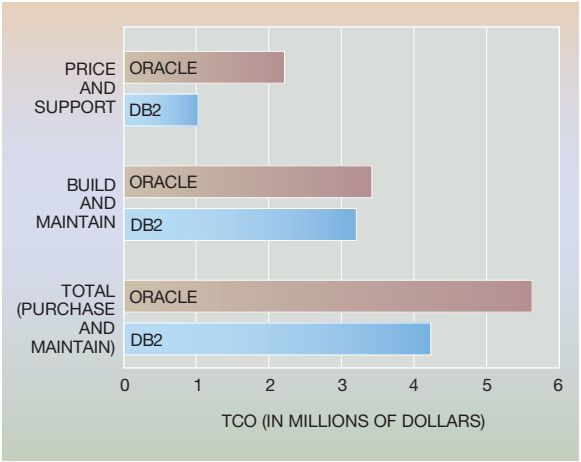
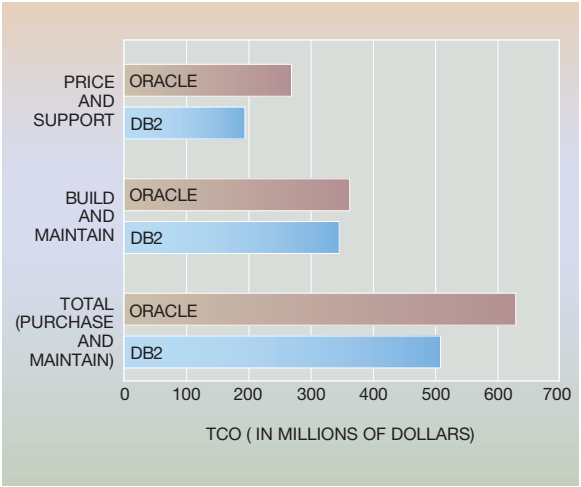


Figure 4 Results of D. H. Brown TCO study for OLTP

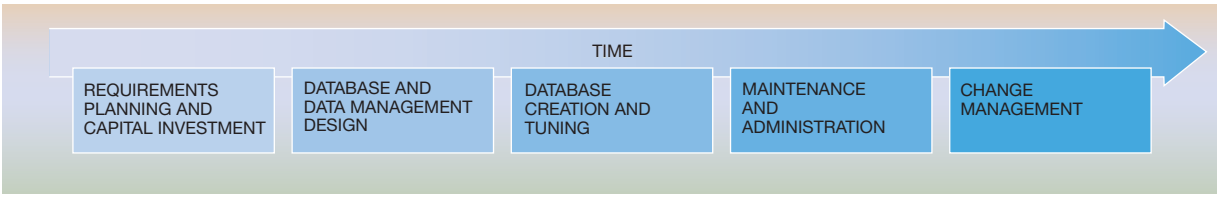


Why relational databases require ease of use

Skilled database administrators and application developers have become increasingly rare, and, following the law of supply and demand, increasingly expensive. A 2001 report from D. H. Brown Associates[6] that compared two leading database products, for both data-warehouse and online-transaction-processing (OLTP) applications, found that a significant portion of the TCO was represented by human administration costs. As illustrated in Figures 3 and 4, the cost for purchasing and support (including the cost of the product licensing) is significantly less than the "build and maintain" cost (which includes the installation, deployment, and ongoing administration of the product.)

Given that human costs have been shown to dominate the TCO of complex IT systems, it is reasonable to ask what keeps the administrators of such systems so busy. In general, the significant advances in database functionality and the burgeoning requirements for database size, connectivity, availability, and heterogeneity increase the administrative burden. For example, data warehouses containing tens of terabytes of data are not uncommon. Popular applications such as SAP (Systems, Applications, Products in Data Processing), which typically create more than 20 000 tables, can be used to support thousands of users simultaneously. In addition, administrators grapple with complex decisions about hardware platforms; shared-nothing, shared-everything, or SMP (symmetric multiprocessing) cluster topology; schema design; constraints and referential integrity; primary keys and indexes; materialized views; and the allocation of tables to disks. Once designed, databases require substantial human input to build,

Figure 5 Time line of database administration



configure, test, tune, and operate. Such ongoing administrative tasks include, but are not limited to, table reorganization, data statistics collection, backup control, security administration, disaster-recovery planning, configuration and performance tuning, and problem analysis.

In this section we describe briefly the scope of tasks for a database administrator, providing a very cursory view which serves to illustrate the large scope of tasks and responsibilities incurred today with typical RDBMS (relational database management system) products. The scope of tasks is easiest to view when imagined along a time line, as shown in Figure 5.

During the initial requirements planning and capital investment, the database designers must arrive at a rough estimate of the performance and storage requirements for the system and select product purchases that will support these requirements. This includes the selection of the RDBMS and of server and storage devices, a selection process often referred to as "capacity planning." During the second stage of system development, the designers concentrate on the logical and physical design of the database (table layout, normalization, referential integrity, indexes and materialized views, triggers, etc.), as well as the overall process strategy for high availability and disaster recovery, data distribution, security, and user management.

During the third stage of development, the database is created, populated, and tuned. Generally, there is a substantial period of testing to validate the operation of the new system with applications and to ensure integration with other systems and operational processes. During the fourth stage, the system goes into production. At this point, extensive involvement is needed by human operators to monitor system operational health, perform query tuning, maintain data statistics, de-cluster/fragment data, maintain storage systems, attend to system repairs and outages, and modify the system design and configuration to account for new operational requirements or to respond to increasing storage needs. Handling system recovery may require management of periodic backup and archival data. In large distributed systems, data replication (cloning) across systems and data integrity checking are common tasks. Moreover, almost all of the design and setup operations performed in stages two and three as described previously may need to be revisited during operation to account for new requirements, data growth, or poor performance. Many systems require complex extra-database operations for data extraction, transformation, and loading and for data replication; these operations also require special tuning and management.

During the fifth stage, the logical or physical design of a database may need to adjust for changing application or usage needs. This can require schema changes and changes to system design and implementation as defined in phases two and three.

In large modern RDBMSs, the tasks described here can be daunting responsibilities as database sizes grow to sizes in terabytes, on systems containing hundreds of CPUs, thousands of storage spindles, and tens of thousands of database storage objects (relational tables and their associated access structures, including indexes, materialized views, and system catalogs).

Administrators of enterprise-class IT systems face considerable challenges to keep their complex, heterogeneous systems up and running. Certainly, the application of autonomic technology has the potential to greatly reduce the administrative effort involved, to eliminate much of its complexity, and to reduce human involvement in the day-to-day drudgery of maintenance, problem determination and resolution, and performance tuning, thereby allowing more time for tasks such as designing and planning.

Given the undeniable fact that traditional IT systems won't evolve overnight into advanced, autonomic systems, there is much value in understanding where administrators could benefit most from automation and "smarts." Identifying the "pain points" where much effort is expended can help indicate where the application of automation and expert assistance can provide the most benefit. Database administrators can be polled as to which tasks are most difficult to execute, which are executed most frequently, and which are most time-consuming.

User-Centered Design approach

In 1994, at the advent of the development cycle for DB2 UDB Version 5, IBM began a major effort in usability analysis for relational databases, focused on User-Centered Design (UCD). Prior to this effort, the efforts of the DB2 UDB design team had focused on the traditional bailiwicks of relational database technology: reliability, performance, and interoperability.[7] Given the focus of enterprises on the total cost of ownership and quick return on investment, the DB2 management team realized that being successful in the marketplace required products that are usable, easy to learn, and easy to maintain. Thus DB2 UDB Version 5 became the first version of the product that used the IBM User-Centered Design approach[8] in a comprehensive and consistent manner, involving multiple disciplines and development teams.

With the help of the User-Centered Design teams in the Toronto and Silicon Valley IBM labs, the UCD process provided the ability to tightly integrate the DB2 design and development work with vigorous gathering of user feedback, extensive iterative testing of design alternatives, thorough evaluation of prototypes and early code, and validation of the completed designs. The UCD involvement was significant throughout the entire development cycle, starting with understanding the users and their requirements and continuing long after the beta versions of the code and its general release.

The results of the increased UCD investment for DB2 UDB Version 5 were extremely encouraging.[7] The successful achievement of the usability objectives set at the beginning of the project and the vigorous UCD involvement contributed to the overall success of the product. The increased ease of use was noted and made reference to by numerous press analysts, and resulted in increased satisfaction ratings from end users. The commercial success of the release proved the effectiveness of the adopted approach and provided a base for continuing the usability focus in subsequent releases.

Database administration requirement survey. During the summer of 2000, IBM ran an extended online survey of over 120 companies to understand their database administration requirements. The survey initially targeted numerous companies with single or heterogeneous database environments. Users with varied database and operating-system experience filled in the questionnaire, which was published on the Internet and distributed via targeted e-mails. The survey was further expanded to include key database applications by independent software vendors such as SAP, Siebel, PeopleSoft, i2, Ariba, and Kana, whose application-development and support perspective brought additional understanding of the key challenges that autonomic computing needed to answer. The majority of the participants were from the United States, but responses also came from Canada, the United Kingdom, Germany, France, Japan, and several other countries. This kind of user-generated input has allowed the designers of DB2 to continue its focus on user-oriented priorities and to clearly reflect this focus in the staging of autonomic manageability features for ease of use.

The results of the survey helped direct the research and development priorities for the autonomic computing aspects of DB2 UDB. The team objective was to deliver enhancements and new features supporting the most important user tasks as determined by the survey, while reducing requirements for time and expertise. This in turn would lead to an increase in usability and significantly reduce the total cost of ownership. As a result of the survey analysis and with some consideration given to the strategic directions of our products, the database group focused a usability effort around three primary administrative areas:

- System health diagnosis
- Query and database tuning
- Maintenance and recovery environments

Scenario development and use. The key tasks in the aforementioned areas were used to develop scenarios that were critical in the design and development of the product. Based on a series of phone interviews with participants in the survey and scenario modeling sessions at the Toronto UCD lab in the IBM Toronto lab, the UCD team developed a number of scenarios focused on the key areas for autonomic

computing. The scenarios underwent an extensive review process by all members of the UCD team. After consensus was reached, the project leads approved the scenarios, enabling the start of the design work. Subsequent to this, changes to the scenarios were made through a carefully monitored and reviewed change request process, which ensured that the changes were in line with the accepted objectives and did not compromise the usability of the tools.

DB2 UDB Version 8 was the first release to use scenarios as a fundamental tool in all stages of the planning, design, and development work, as follows:

Planning: Scenarios were used to bring forward ideas for new functionality and tools and help the management team to prioritize.

Competitive Evaluation: Scenarios were used to identify major strengths and weaknesses of competitive products.

Design: Scenarios helped in the conception and prototyping of design alternatives.

Iterative Design Explorations: Scenarios were used as a base for discussions with participants in user feedback sessions and helped in the selection of the best options among design alternatives.

Design Evaluation and Validation: Participants were asked to complete scenarios with early code drivers to evaluate or validate designs and help identify potential usability issues.

User adoption and acceptance of autonomic administration. While it is technically possible to automate many database administration tasks, there remains much that currently requires at least some human intervention in the area of problem resolution. Identifying and resolving problems within a database can be a complex and difficult process requiring skills and experience that are sometimes beyond those of a part-time database administrator (DBA). Our questions to both the experienced and part-time DBAs concerned how much autonomic database administration would be acceptable to them in the area of problem resolution. Their answers showed there was a difference of opinion between those who work full-time on a database and those who maintain a dual role such as a DBA who is also an application developer.

We conducted numerous sessions with users of databases from all of the major vendors. These sessions comprised focus groups, scenario generation sessions, interactive design sessions, and, later, software evaluation and validation sessions. Through these sessions, we could see a distinct pattern regarding the acceptance of automated problem resolution in the database. When the concept of automating system-generated recommendations was raised with participants, the two groups (full and part-time DBAs) diverged. Both user groups welcomed the concept of providing recommendations (with the less experienced users being particularly enthusiastic).

How much autonomic database administration would be acceptable for problem resolution?

Those who work full-time on databases are typically highly skilled database users and rely heavily on their previous experience to solve problems. When full-time users were questioned about the level of database automation that they were willing to accept, they typically opted for very low levels. Full-time DBAs were not willing to trust decisions that could affect the performance of their database to an automated solution. They were, however, interested in the system's ability to offer recommendations. The experienced users wanted to view the recommended commands in detail and have the ability to directly edit them before their execution.

Full-time DBAs differed from their part-time counterparts in that they were much more concerned with the detailed content of the recommendations and were very much averse to the idea of complete automation. Full-time DBAs are generally employed in companies where database performance is imperative and any performance drop can have large effects upon scheduling and work flow. There were very few scenarios where experienced full-time DBAs would accept an automated solution to a problem; they preferred, instead, to be notified and provided with recommendations.

Database users who did not have a high level of experience (part-time DBAs tend to fit into this group) differed significantly from those who work on a da-

tabase on a full-time basis. A part-time DBA tends to come from a non-database-oriented background. The most common combination we encountered was a DBA who was also a developer. Developers tend to treat the database purely as a storage resource and are not particularly concerned with optimizing performance; instead, their primary concern is keeping the database running at overall speed levels that do not impair application performance. Troubleshooting is handled on an "as needed" basis, and performance tuning, often considered only as an afterthought, becomes a priority only if there are significant performance issues filtering up to the application level.

Part-time database users were significantly more interested in database automation capabilities than their full-time counterparts. The part-time DBAs were happy to accept system-generated recommendations; these DBAs were willing to take them at face value and did not require a complex explanation of the proposed commands.

The reason for acceptance or non-acceptance of the automation of recommendations lay in whether the participants were willing to trust a machine to provide safe recommendations. Experienced DBAs were reluctant to believe that a machine could understand their systems and provide recommendations that were intelligent enough to counteract the problems, due to the complex nature of their environments and concerns over the consequences of errors. Part-time DBAs were very willing to trust the system to provide accurate recommendations and were happy to have a piece of work (that is often not central to their activities) taken off of their hands, even if this meant the system was not optimally tuned. Part-time DBAs held the opinion that they would trust recommendations until they found them to be incorrect.

These findings show that for an autonomic solution to be accepted by both part-time and full-time administrators, user trust must be addressed with at least as much care as the autonomic solution itself, raising potential issues for the future, such as the offering of varied levels of automation.

Autonomic computing and DB2 Universal Database Version 8

The DB2 autonomic computing project is clearly a step in the direction of providing an advanced autonomic database product. This section describes the major elements of DB2 UDB Version 8 that incorporated autonomic computing functionality.

Design considerations. Our objective was to automate challenging and difficult tasks and to make the database self-managing and self-tuning—ensuring constant availability and optimal performance. However, this does not mean that the need for database administrators will be eliminated, at least not in the near future.

The feedback during the course of the project proved that the changes introduced by autonomic computing will have a significant impact on the requirements for knowledge and expertise, in particular for resolving database health issues. By automating many of the time-consuming tasks, DBAs will be made more efficient. At the same time, the participants clearly proved that complex environments need their attention and their judgment before an action is taken.

Advanced DBAs have significant responsibilities related to the availability and the performance of the databases. The dynamic business environment today requires that application end users have instant access to data. Millions of users are browsing the Web for information that resides in databases, and unavailable data would affect their satisfaction and jeopardize a company's ability to generate revenue or get the attention of potential clients.

DBAs need to understand the details of what is happening in their environment and what the ramifications of a database change are from a business perspective. This understanding enables them to make changes based on the business requirements for the particular enterprise. Therefore, for a database to be autonomic, it needs to be able to "think" not only in technical, but also in business terms. This is the next goal of the DB2 autonomic-computing initiative and will require further work with users, using contextual inquiries and building conceptual models of the different business environments and their relation to the technical capabilities and configuration of the database product.

Management by exception and Health Monitor. Monitoring the health of the database system was clearly identified as a very important feature for the users of the database. DBAs need to do this on a constant basis, and it requires extensive understanding of the system parameters, configuration, and resource utilization. Advanced knowledge is required to react to a problem identified through monitoring

Table 1 Usability benefits of the management-by-exception model (Part A)

Health Monitoring before DB2 UDB Version 8	Health Monitoring in DB2 UDB Version 8
Determine which database system parameters need to be monitored.	Low-cost automatic health monitoring of important database system parameters begins immediately after installation.
Determine which database objects will be monitored.	All DB2 Version 8 instances and databases are monitored with a low performance impact (less than 1 percent).
Identify/create a snapshot of event monitors needed to collect data for the health of the database system.	The Health Monitor is available to the user immediately after installation.
Determine the threshold values that indicate health problems for the monitored parameters.	Set of default system parameters to be monitored, based on study of typical database environments, is specified at installation.
Implement notification mechanisms and start monitors.	The Health Center notifies users of issues by means of a DB2 message or an animated status icon (Health Center Beacon, available in the DB2 GUI tools). Notification by e-mail is also available and can be configured after installation.

Table 1 Usability benefits of the management-by-exception model (Part B)

Health Issue Management before DB2 UDB Version 8	Health Issue Management Using the Health Center in DB2 UDB Version 8
Identify an issue from the performance or event monitor data.	The Health Center registers potential health issues and notifies the user.
Find more details and review them—monitor and "drill down" in the available information further.	The details of the alert are provided in the Health Center.
Determine one or more alternative actions to fix the problem (use help information and manuals to do this), and determine how other parameters will be affected.	The Health Center provides one or more recommendations for resolving the issue.
Write scripts (if necessary) to execute the actions or invoke the needed tools.	Click one button to execute the recommendations.
Verify that the problem has been resolved (by running a script or checking the affected objects).	The Health Center confirms the result of the action. Health monitoring continues, and after a refresh occurs, the resolved issues are removed from the alerts, confirming the resolution.

or end-user reporting—the database administrator needs to assemble more detailed information and determine what steps need to be taken to resolve the issue. All of this happens in a time-restricted environment, requiring the DBA to work under pressure and deliver the resolution in a timely manner.

With this in mind, the DB2 autonomic computing team focused on providing DBAs the ability to manage the system health of the database without the need for constant monitoring and helping them to resolve any potential issues proactively and quickly. This management-by-exception approach, wherein system health is monitored and managed automatically unless the DBA's attention is required, represents a paradigm shift away from the historical method of DBA polling. System health monitoring has been implemented by the development of a reliable autonomic health analysis mechanism, which detects problems without user intervention and notifies administrators via e-mail, pager, or other means. The new Health Center GUI (graphic user interface) tool has been introduced, providing tools for the detailed analysis of problems that have been autonomically sensed. It should also be noted that the supported management-by-exception model provides the user with the ability to periodically poll the system status.

The key characteristics of the management-by-exception approach in DB2 UDB Version 8 are:

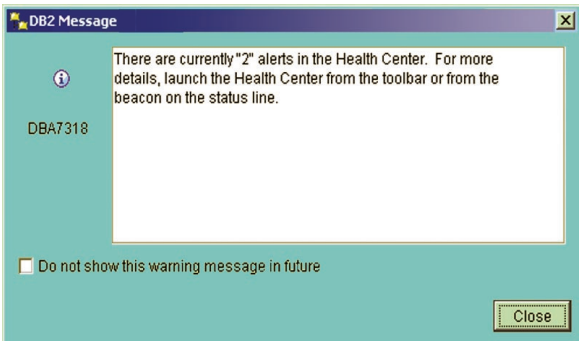
- DB2 monitors the health of the database system by default. It measures key health indicators and notifies the DBA when there is a potential health issue that needs attention.
- The Health Monitor is available to run immediately after installation. No additional configuration is necessary.
- The Health Center provides details about the health issue and suggests one or more actions that can be taken to resolve or prevent the problem. Thus, the user, even the novice DBA, can maintain a healthy environment and high performance without the need to seek expert help or consult extensive documentation (as with some of the existing database products).
- The functionality of the Health Center is available for users who prefer working from a command line interface (a significant number of DBAs).
- This functionality is available for the mobile user as well. The DB2 Web Health Center and a PDA version of the Health Center are available in Version 8.

Our objective in implementing the management-by-exception model was to enable even less experienced users to maintain a healthy and high-performing database environment. At the same time, the tool had to be useful for experienced users as well by allowing them to increase the efficiency of their work, by allowing them to focus on objects that need attention (rather than trying to identify which objects need attention), and by allowing them to react quickly to the challenges of the complex environments in which they work.

Table 1 (Parts A and B) clearly demonstrates the benefits of the management-by-exception model in system health monitoring compared to the traditional approach. Part A of the table shows the usability benefits in setting up health monitoring, and Part B does the same for reacting to a health issue. With the new model, most of the tasks that require expertise and involvement from the DBA have been automated. In addition, this model serves as a basis for completely automating the tasks of monitoring and problem resolution.

A key element of the management-by-exception model is getting the DBA's attention when it is necessary. Indeed, the objective of the Health Center is to be invisible unless there is something that needs attention. After the Health Monitor discovers an issue, it notifies the DBA via a DB2 message (see Figure 6), pointing the user to the Health Center. Alternatively, an animated icon, the Health Center Beacon, has been integrated with the rest of the DB2 GUI tools. If there is a new alert in the Health Center, the Beacon flashes to get the user's attention. The user can invoke the Health Center by clicking on the icon. Finally, e-mail notification functionality is available to the user, in which a message is sent to the selected addresses (which could be pagers as well), to alert the user that there is a new issue.

Figure 6 Health Monitor message



To help the user determine the severity of the particular alert, the Health Center categorizes the issue as a warning, alarm, or attention, based on different thresholds. These thresholds, like the notification options, are configurable. The thresholds all have default values, allowing even the novice user to keep the database healthy.

A key function of the Health Center is to provide recommendations for resolving health issues. Indeed, while its functionality automates most of the health monitoring tasks, the autonomic functionality lies in knowing what needs to be done in order to return the system to a healthy state. To determine the recommendations for resolving issues related to the different health indicators, the team relied on the expertise of numerous internal experts from DB2 development, DB2 support, and DB2 services. After the set of recommendations was determined, the UCD team ran a series of user feedback sessions with internal users to evaluate the efficiency of the proposed actions.

To complement the standard Health Center, a Web-based version was also included, to facilitate remote administration. The benefits of this interface can be exploited by remote workers, who can view the health of the databases for which they are responsible and apply recommendations to resolve problems without having to come into the office.

Query optimizer. Query optimizers are one of the most autonomic features of today's relational database systems, automatically determining the best way to execute a declarative SQL (Structured Query Language) query. Since its inception, DB2's query optimizer has automatically optimized even the most complex decision-support queries—without any of the "hints" from the user required by some competitors' optimizers. It performs this optimization using a combination of: (1) powerful query rewrite rules to transform queries written by the user (or, more commonly, a query generator) into standardized, easier-to-optimize queries,[9,10] and (2) a detailed cost model to generate and evaluate a rich set of alternative plans for executing the query.

The optimizer automatically determines whether any existing Materialized Query Tables (MQTs, i.e., materialized views) could benefit a query, and if so, "routes" the query to use the MQT without having to alter the query in the user's application program. It collects statistics on the size of each table and the distribution of each column to model how many rows must be processed by any query a user might submit. It adapts its model to the environment in which it is optimizing, automatically factoring in the speed of the CPU, the storage devices, and the network-connecting machine clusters (in a shared-nothing environment) and/or sites (in a federated environment). In most cases, the optimizer minimizes the total overall resource consumption; in parallel environments, it automatically uses the minimal elapsed time as the optimization criterion.

The cost model includes detailed modeling of the availability of various memory categories (multiple buffer pools, sort heap, etc.) versus the amount needed, hit ratios, the cost to build temporary tables versus the cost to re-scan them, various flavors of prefetching and big-block I/O, non-uniformity of data distributions, and so forth.[11]

Configuration Advisor. This component configures the major memory areas of the database, as a system configuration task. The configuration of a database system is critical to system performance, as it includes allocation of system memory for major database operations, such as data caching, sorting, and networking. Database configuration also defines a number of database operational parameters, such as the number of database server agents, I/O subagents, logging frequency, and so forth. The advisor configures over 35 configuration parameters. To do this, the advisor is designed to evaluate the setting of each configuration parameter based on characteristics of the database system. The characteristics used in the database model include system environment data, which the advisor senses automatically (including the size of the system RAM, number of storage disks, and number of CPUs), and data specified by the user. The gathering of user-specified information is specifically designed with the assumption that the user has a very low skill level. The combined set of characteristics is then used to derive the value of each configuration parameter as a weighted function of the system characteristics.

Allocation of the system memory to the memory-consuming components (data cache, sort, network memory, etc.) is assumed to be a zero sum process, and therefore the values of these parameters are determined in a combined model, taking careful account of each component's requirements, the database system architecture, and the available system memory.

The advisor can be invoked in DB2 UDB through either a GUI or a programmable API. A number of commercial database applications that use DB2 for their relational store invoke this advisor through the programmable API.

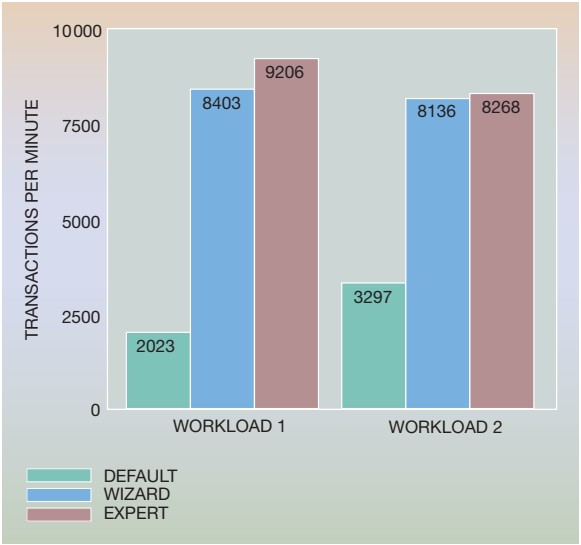
Research and development continues on the Configuration Advisor, enhancing its modeling for a future release of DB2. Recent experiments with the remodeled algorithms have shown dramatic results, particularly with OLTP and batch database systems. Figure 7 shows the results of two experiments using an industry standard OLTP benchmark. The experiments were performed on two distinct servers. The diagram illustrates for each experiment how the system throughput was improved over the default settings after running Configuration Advisor. The throughput was then compared to the performance achieved by a human expert, who was given an extended period of several days to adjust the database configuration for improved performance. In the first experiment, the Configuration Advisor achieved 91.3 percent of the throughput performance of the same system tuned by an expert. During the second ex-

periment, the Configuration Advisor achieved 98.4 percent of the throughput of the expert-tuned system. In both cases, the Configuration Advisor outperformed the default settings significantly. These early results suggest that autonomic performance configuration is an achievable goal in the near term for an important class of database workloads.[4]

Design Advisor. Database designers grapple with several physical database design problems, relating to how the data is stored and accessed. Some of these physical design problems include: selection of fast lookup indexes, materialized view selection, storage topology, clustering keys, and partitioning keys. Determining the optimal set of indexes to create has been a long-standing database research problem and the topic of numerous papers over the past two decades. The DB2 Design Advisor, which has been part of DB2 since Version 6 (released in 1999), aids physical database design by recommending indexes for tables through analysis of a specific workload for one or more SQL statements (including INSERTs, UPDATEs, and DELETEs). The workload may be automatically captured or supplied by the user.[5] The Design Advisor exploits the detailed performance model of the query optimizer to evaluate the potential benefit of virtual (simulated) candidate indexes for the target workload. Using the database's internal cost model not only allows DB2 to evaluate the potential cost benefit of each virtual index, but also provides some reasonable assurance that the newly recommended indexes are likely to be selected by the database during access plan selection. Current research is extending the Design Advisor to provide recommendations on a number of addition physical design problems.

Automatic specification of query parallelism. DB2 can automatically determine at run-time the most effective degree of query parallelism to use to improve query performance across SMP CPUs as a maintenance task. Parallel access can prove inefficient for short-duration operations by adding more overhead in context switching and communication costs than benefits. Automatic specification of parallelism means that during execution complex queries can benefit from parallel processing, while simple queries can bypass the overhead of the parallel-processing infrastructure. The decision on the degree of parallelism can be made dynamically during execution. This dynamic ability to determine a near-optimal degree of parallelism for query execution makes much of the past literature on load-balancing obsolete.

Figure 7 Configuration Advisor performance results



Load utility automatic tuning. The DB2 load utility performs mass insertion of data into a specified target table. To do so, it exploits a series of concurrent (parallel) subagents for data prefetching, formatting, and direct writing to database system storage. The efficiency of the load process is heavily dependent on whether optimal selections are made for memory consumption (used for buffering and sorting of data), the number of parallel formatting subagents, and the number of I/O subagents. The load utility removes this burden from the user by automatically selecting the degree of memory consumption, I/O parallelism, and SMP parallelism. This is accomplished by examining the table characteristics, memory free space, the number of table space containers (virtual storage devices), and the number of system CPUs online.

In addition, the load utility maintains table indexes defined for the target table. These index structures can be maintained in one of two ways, either by completely rebuilding them or by incrementally extending them with the new data tuples. A tuple consists of a number of values separated by commas, for instance: S = {12345, 6789, 'hi'}. The choice of maintenance technique is not trivial, given that the per-tuple maintenance cost is generally far more expensive during incremental index maintenance. The load utility selects the appropriate maintenance mode automatically during execution, based on an

analysis of the index structure complexity and the ratio of newly loaded data to existing table data.

Query Patroller. The DB2 Query Patroller acts as a gate-keeper for DB2, accepting, analyzing, prioritizing, and scheduling database requests and (optionally) notifying users when their requests have been processed. Guided by policies established by the user in a profile, the Query Patroller limits bursts of arrivals or long-running queries to the server, preventing its saturation and ensuring sufficient resources for queries that are executing.

The Query Patroller first determines the relative cost to execute each query, using the cost estimate provided by DB2's Explain facility, which exploits a complex cost model to estimate the resource consumption for the access plan of a given query. Note that the Explain facility is a model and does not actually execute the specified SQL statements. It then uses this estimated cost to determine when the query should be run. If the cost exceeds a threshold established by the user's profile, the query is held for manual intervention by the system administrator, and the user is notified. Otherwise, Query Patroller schedules the query for execution by an agent, taking into account: (1) the current number of queries executing on the system, (2) the cost of all queries currently executing, (3) the number of nodes in the system, (4) individual user priorities, and (5) the number of queries executing for each user. After a query has completed execution, the user is notified via e-mail and, if the job accounting status is active, a row is added to the job accounting table. Information in this table is used to provide reports and display database usage history.

DB2 uses a patented technique to automatically protect the integrity of the data by ensuring that DB2 detects any corrupted data from incomplete I/Os when it reads the disk. The method exploits consistency bits to verify that a page being read into the buffer pool from disk is neither a "partial page" nor has it been changed by some form of disk corruption.

Consistency bits were introduced in DB2 Version 2. A bit from each sector of storage on a page is set to the same value before writing the page. When the page is read, the DB2 Data Manager verifies that all of the bits are the same. If some of the bits are different, it indicates a partial page write or disk corruption. The net result is continual automatic validation of storage consistency as pages are read from disk by the DBMS.

Conclusions and future work

Autonomic computing offers a fresh approach toward ease of use, an approach focused on self-management rather than on the simplicity of the interface. The autonomic computing work for DB2 has included a focus on User-Centered Design, and has resulted in the development of a number of powerful usability enhancements for administrators. The latest of these features support system integrity assurance, physical database design, and database tuning. A number of additional features currently under research and development will expand autonomic capabilities for physical database design, advanced problem determination, and system tuning.

*Trademark or registered trademark of International Business Machines Corporation.

Cited references and notes

1. *Living in an On Demand World*, IBM Corporation (2002), http://www.ibm.com/ebusiness.
2. A. Ganek, *A Letter from Alan Ganek,* http://www.ibm.com/autonomic/letter.html.
3. D. M. Russell, P. Maglio, R. Dordick, and C. Neti, "Dealing with Ghosts: Managing the User Experience of Autonomic Computing," *IBM Systems Journal* **42**, No. 1, 177–188 (2003).
4. D. Patterson, "ISTORE: A Server for the Post-PC Era," Presented September 25, 2000, IBM Thomas J. Watson Research Center, New York.
5. Sun Microsystems has an initiative known as "N1," which is focused on automating the data center. HP has a similar initiative known as "UDC" or "Utility Data Center."
6. IBM DB2 Universal Database V8.1 vs. Oracle9iR2: Total Cost of Ownership, D. H. Brown Associates Inc. (November 2002), http://www-3.ibm.com/software/data/pubs/pdfs/dhbrown.pdf.
7. R. Sobiesiak, B. Jones, and S. Lewis, "DB2 Universal Database: A Case Study of a Successful User Centered Design Program," *International Journal of Human-Computer Interaction* **14**, Nos. 3–4, 279–306 (2002).
8. K. Vredenburg, S. Isensee, and C. Righi, *User-Centered Design: An Integrated Approach*, Prentice Hall, New Jersey (2002).
9. H. Pirahesh, J. M. Hellerstein, and W. Hasan, "Extensible/Rule Based Query Rewrite Optimization in Starburst," *Proceedings of the 1992 ACM SIGMOD Conference*, ACM, New York (1992), pp. 39–48.
10. H. Pirahesh, T. Y. C. Leung, and W. Hasan, "A Rule Engine for Query Transformation in Starburst and IBM DB2 C/S DBMS," *Proceedings of the 1997 IEEE International Conference on Data Engineering*, IEEE Press, New York (1997), pp. 391–400.
11. P. Gassner, G. M. Lohman, K. B. Schiefer, and Y. Wang, "Query Optimization in the IBM DB2 Family," *IEEE Data Engineering Bulletin* **16**, No. 4, 4–18 (1993).

Accepted for publication May 30, 2003.

Ric Telford *IBM Autonomic Computing, 4205 Miami Blvd., Research Triangle Park, NC 27709 (rtelford@us.ibm.com).* Mr. Telford is the Director of Architecture and Technology for IBM's autonomic computing initiative. He received his B.S. degree in computer science from Trinity University in San Antonio, Texas, where he graduated Phi Beta Kappa. He subsequently joined IBM in Dallas, Texas where he worked on office systems development. Mr. Telford has managed a number of software technology areas throughout his career at IBM, including networking software, digital imaging, security, mobility software, directory services, and business integration. He also served as Technology Director for the IBM CIO, delivering technology solutions for the IBM intranet.

Randy W. Horman *IBM Data Management Division, IBM Toronto Lab, 8200 Warden Ave, Markham, Ontario, L6G 1C7 (horman@ca.ibm.com).* Mr. Horman is a Senior Technical Staff Member on the DB2 development team at the IBM Toronto Lab. He received a B.A. degree in mathematics, computer science, and economics, as well as an M.Math degree in computer science from the University of Waterloo in 1994 and 1995, respectively. He subsequently joined IBM at the Toronto Lab, where he began working on the parallel database system, DB2 Parallel Edition. Recently, Mr. Horman has focused his attention on database manageability, and in particular, the applicability of autonomic technology. Mr. Horman is a member of the Association for Computing Machinery and the Computer Society of the Institute of Electrical and Electronics Engineers.

Sam Lightstone *IBM Data Management Division, IBM Toronto Lab, 8200 Warden Ave, Markham, Ontario, L6G 1C7 (light@ca.ibm.com).* Mr. Lightstone is a senior technical development manager with IBM's DB2 Universal Database development team within the IBM Data Management Division. He leads the autonomic computing research and development effort for DB2 and is a member of IBM's Autonomic Computing Architecture Board. His current research includes numerous topics in autonomic computing and RDBMSs. Prior to his current position, Mr. Lightstone worked as development manager and technical lead for the DB2 load-and-sort development components and was overall project manager for the first production-ready shipment of the DB2 UDB version 7.1 release. Mr. Lightstone has published and lectured on topics including autonomic computing, database systems, voice encoding, image processing, object-oriented design, intellectual property, and software testing and speaks frequently at DB2 conferences. He has been with IBM for eleven years.

Nikolay Markov *IBM Data Management Division, IBM Toronto Lab, 8200 Warden Ave, Markham, Ontario, L6G 1C7 (nikolay@ca.ibm.com).* Mr. Markov holds M. Sc. degrees in computer science from the University of Sofia (Bulgaria) and in human-computer interaction from Heriot-Watt University in Edinburgh, Scotland. He joined the IBM User-Centered Design team at the IBM Toronto Lab in 1995, where he initially worked on gathering and analyzing user feedback for the DB2 UDB Version 5. His work also included competitive analyses and evaluations of competitive DBMS products. More recently, Mr. Markov has focused on IBM-wide initiatives like the autonomic database and improving the database for small and medium size organizations.

Stephen O'Connell *IBM Data Management Division, IBM Toronto Lab, 8200 Warden Ave, Markham, Ontario, L6G 1C7 (soconnel@ca.ibm.com).* Mr. O'Connell holds a B.S. degree in industrial design and technology from Loughborough University (1997) and an M.Sc. degree in work design and ergonomics from Birmingham University (1998) where his thesis (working with British Telecom) examined multimodal interfaces in the context of future communication devices. At EDS (1998–2000), he worked on the software design and deployment of call centers for the British government. After working with Nortel (2000–2001) on an integrated voice, e-mail and fax product, he joined the IBM Toronto Lab in the DB2 User Centered Design team. While at IBM, Mr. O'Connell has worked on projects including Web clients, DB2 GUI tools, cross-product problem resolution, and IBM-wide-initiatives including the autonomic computing initiative.

Guy M. Lohman *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (lohman@almaden.ibm.com).* Dr. Lohman is manager of Advanced Optimization in the Advanced Database Solutions Department at the Almaden Research Center in San Jose, California, and has 21 years of experience in relational query optimization. He is the architect of the Optimizer of the DB2 Universal Database (UDB) for Linux®, UNIX®, and Microsoft Windows®, and was responsible for its development in Versions 2 and 5. During that period, Dr. Lohman also managed the overall effort to incorporate into the DB2 UDB product the Starburst compiler technology that was prototyped at the Almaden Research Center. More recently, he was a co-inventor and designer of the DB2 Index Advisor, and co-founder of the DB2 SMART (Self-Managing and Resource Tuning) project, part of IBM's autonomic computing initiative. In 2002, Dr. Lohman was elected to the IBM Academy of Technology. His current research interests involve query optimization and self-managing database systems.