Towards Workload Shift Detection and Prediction for Autonomic Databases

Marc Holze
University of Hamburg
Vogt-Koelln-Strasse 30
Hamburg, Germany
holze@informatik.uni-hamburg.de

Norbert Ritter
University of Hamburg
Vogt-Koelln-Strasse 30
Hamburg, Germany
ritter@informatik.uni-hamburg.de

ABSTRACT

Due to the complexity of industry-scale database systems, the total cost of ownership for these systems is no longer dominated by hardware and software, but by administration expenses. Autonomic databases intend to reduce these costs by providing self-management features. Existing approaches towards this goal are supportive advisors for the database administrator and feedback control loops for online monitoring, analysis and re-configuration. But while advisors are too resource-consuming for continuous operation, feedback control loops suffer from overreaction, oscillation and interference.

In this position paper we give a general analysis of the parameters that affect the self-management of a database. Out of these parameters, we identify that the workload has major influence on both physical design of data and DBMS configuration. Hence, we propose to employ a workload model for light-weight, continuous workload monitoring and analysis. This model can be used for the identification and prediction of significant workload shifts, which require autonomic re-configuration of the database.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems

General Terms

Management

Keywords

Autonomic Databases, Workload Modelling, Workload Shift Detection, Workload Shift Prediction, DBMS Architecture

1. INTRODUCTION

At its core, a database system (DBS) is supposed to be a reliable, data independent storage system. With the advance of hardware and software development, the demands on these systems have increased significantly. Database vendors have reacted by adding new features to their systems with every release, using them as a selling pitch. As a consequence, today's database management systems (DBMS) have become overloaded with features. The result of this featurism is an increasingly complex architecture of commercial DBMSs. At the same time, there has been considerably less attention on the development of administration interfaces and manageability. For these reasons, the administration of DBSs today is a challenging task, which can only be performed by highly-skilled, scarce and thus expensive database administrators (DBA). Hence, it has been noticed ([3], [7], [13]) that the total cost of ownership of a DBS nowadays is no longer dominated by hardware cost, but by "people cost" for skilled application developers, DBAs, and their training on new product versions.

Autonomic, i.e. self-managing, databases are considered as the way out of the problem of increasing database maintenance costs. The doctoral work outlined in this position paper focuses on the analysis of tasks that are related to self-management of a DBS. In particular, it investigates the workload-dependency of the self-management functionality. In the remainder of this section, we introduce the features expected from autonomic databases, and discuss why the current approaches fail to reach this goal. Afterwards, we shortly sketch our idea of workload modelling to efficiently handle the workload-dependency of autonomic databases.

1.1 Autonomic Databases

The basic idea for the reduction of the total cost of ownership is to apply the principles of autonomic computing to database systems. They should be able to maintain themselves to a large extent, or should at least provide concrete recommendations to the DBA when action is required. Currently there are several definitions about the general characteristics of autonomic systems in literature. We agree with [11] and consider the following properties to provide a good definition of the characteristics of autonomic systems: Self-Configuration, Self-Optimization, Self-Healing, and Self-Protection. *Self-Configuration* describes the ability of a system to automatically adapt its configuration to its hardware and software environment. All required changes of configuration parameters (knobs) must not disrupt the operation of the system. Besides adaptation to environmental changes, an autonomic system should also be able to internally optimize its performance (*Self-Optimization*). For this, the system has to continuously monitor its own performance and optimize itself whenever possible. A sys-

PIKM'07, November 9, 2007, Lisboa, Portugal.

tem is *Self-Healing* if it is able to recover from failures. No matter whether the error was caused by internal or external reasons, the system must be able to detect it, analyse it and take appropriate recovery actions. *Self-Protection* of autonomic systems refers to the ability to protect against attacks. This includes the authentication and authorization across all components as well as intrusion detection.

When comparing existing DBMSs with the properties of autonomic systems described above, it is obvious that certain aspects have been implemented for a long time. Query optimization, for example, clearly can be considered as a self-optimizing feature. Other examples are self-healing recovery mechanisms based on logging and backup, and fine-grained read/write privileges on database objects for self-protection. In addition, database vendors have started to enhance their products with an increasing number of *autonomic features*, i.e. supplementary tools and components that increase the autonomy of the DBMS. These autonomic features range from physical design recommendation tools ([3], [7], [21]) and configuration wizards at setup-time ([13]), over runtime-adaptable configuration and tuning parameters ([14], [18]), to advanced automated statistics collection mechanisms ([1]). Detailed surveys of the autonomic features in today's DBMSs can be found in [9] and [19].

1.2 Current Approaches

Although the database industry by now has integrated a large amount of autonomic features into their products, only two general concepts can be identified in the area of autonomic databases:

Advisors are intended as administrative tools to provide the DBA with recommendations for performance improvements by physical design optimization. As such, their analysis is explicitly triggered by the DBA when he suspects the possibility for a significant improvement. Examples for advisors in commercial DBMSs are IBM DB2 Design Advisor [21], Microsoft SQL Server Database Tuning Advisor [3] and Oracle SQL Tuning Advisor [6]. The inputs for these advisors are a typical workload of the DBS, and a set of constraints, e.g. the maximum space for indexes. Using expert knowledge in their heavy-weight analysis algorithms, the advisors determine the set of indexes and materialized views which provide the lowest execution cost for the given workload.

Feedback Control Loops originate from control theory [8]. In the area of autonomic databases they describe a design template for autonomic features, which is illustrated in Figure 1. In a feedback control loop, a *controller* via *sensors* continuously monitors the performance of a specific database component (the *managed resource*) according to a predefined metric. When the sensor data indicates the need for re-tuning, the controller autonomically plans and executes actions to adapt to the new situation by re-configuring the values of the *effectors*, i.e. the DBMS's tuning knobs. As it runs constantly, the monitoring and analysis of feedback control loops must be as lightweight as possible. Examples for the usage of feedback control loops are memory management ([14], [18]), statistics collection([1]), and utility throttling([16]). The usage of feedback control loops in today's DBMSs suffers from three problems: Due to the complex architecture of the DBMSs, the consequences of changes to tuning knobs are often not quantifiable. For this reason the re-configuration decisions bear the risk of *overreaction*. Second, a re-configuration decision of the control loop may lead to a system state as bad as the original one. In this case the problem of *oscillation* between two equally bad configurations may occur. Third, the tuning logic in the control loops is highly specialized to a certain domain. Nevertheless, its tuning decisions may have side-effects on other system components managed by other control loops. Currently there is no concept to manage this *interference* of control loops.

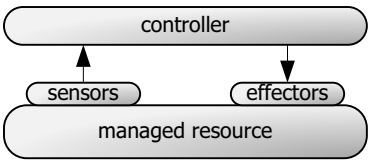

Figure 1: Feedback Control Loop Design Pattern

The problems of overreaction, oscillation, and interference of feedback control loops are caused by the complex architecture of today's DBMSs, which make it impossible to exactly predict the consequences of re-configurations. We describe an approach towards a system design that overcomes these problems in Section 2.

1.3 Database Workload Modelling

Advisors and feedback control loops are used to change the database configuration according to the usage (workload) of the DBS. Regarding real life operation, the workload is determined by the set of applications using the database in an enterprise. Hence, the workload of a database in an enterprise can be assumed to be stable (within certain boundaries) most of the time. And as long as the workload is stable, monitoring for workload changes is unnecessary overhead. Instead, tuning of the physical design and the re-configuration becomes important as soon as there is a significant change in the workload, e.g. because an additional client application has been installed.

To autonomically adapt its configuration to its current usage, the DBMS must perform continuous monitoring and analysis of the workload. In this paper we propose an approach that will drastically reduce the overhead required to achieve this goal. Our approach is based on the idea of reducing the monitoring of the autonomic features of a DBMS to a minimum as long as the workload is stable, and to intensify it when there is a change. In order to identify these changes, we propose to maintain a compact *workload model* that represents the typical composition of the database's workload. Using this model we will be able to quickly identify when the workload significantly differs from the workload in the past, and to react to these shifts by triggering re-configuration. Additionally, the workload model is intended as a means for the proactive optimization of the DBS configuration: by analysing the model for patterns, we will be able to predict periodic shifts in the workload, and to adapt the DBS configuration in advance.

The remainder of this paper is structured as follows: In Section 2 we shortly describe our approach towards the long-term goal of a fully autonomic database. Afterwards, we discuss our ideas about workload modelling in more detail in Section 3, and show how they fit into our fully autonomic database approach. We discuss related work in Section 4, before we describe the roadmap for future work in Section 5. Section 6 summarizes our approach.

2. AUTONOMIC DBMS DESIGN

The implementation of database management systems has been dominated by *layered approaches* for a long time. In the following section we will show that building a fully autonomic DBMS will be almost impossible with this architecture. Afterwards, we roughly sketch our ideas of how DBMSs could be designed to achieve this goal.

2.1 Classic Database Architecture

In layered DBMSs every layer follows the principle of information hiding. Structuring the DBMS into layers provides several advantages, e.g. an easier implementation of higher layers because of a higher level of abstraction, and an increased robustness against implementation changes within the layers. System R for example, has layered its relational data system upon a relational storage system. Later, this architecture has been refined and split into more functional layers. Figure 2 illustrates the five-layer architecture of a DBMS proposed by Härder and Reuter in [12]. But although DBMSs are conceptually structured into multiple layers, some of the layers are often merged for performance reasons. Furthermore, today's systems are shipped as monolithic blocks, and their internal structure is opaque to the customer.

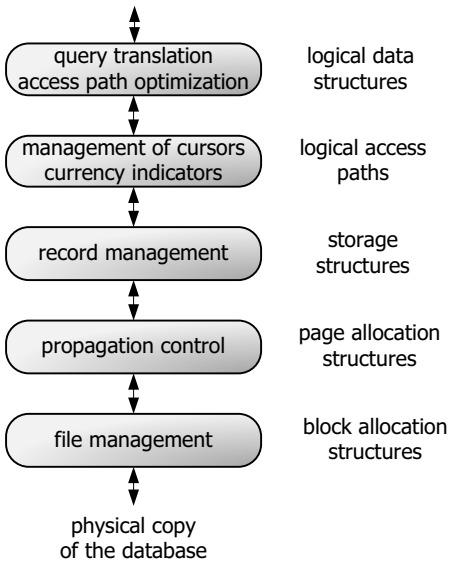

Figure 2: Layered DBMS proposed by [12]

Weikum et al. have shown in [19] that the current DBMS architecture is not suitable for fully autonomic databases. Even in a strictly layered architecture, there are plenty of *implicit interdependencies* between the system components. On the one hand these dependencies are caused by resource sharing between the layers. For example, increasing the page buffer size on the page allocations layer will increase the layer's local performance, whereas it might have negative impact on the logical access path layer because of reduced memory availability in the sort area. On the other hand implicit interdependencies are caused by interferences of the data processing mechanisms across the layers. If for example the optimizer in the logical data structures discovered that an additional index would accelerate query processing, then it is not easy to quantify the additional maintenance costs of this index in e.g. the block allocation structures layer.

Above examples illustrate that the complexity of DBMSs makes it hard to even predict the system-wide effects of changing a single configuration parameter or a physical design decision. Hence, today's feedback control loops monitor and analyse the performance and workload related to one configuration parameter of one system component only. Neither do they consider possible side-effects on other components, nor are they able to coordinate their tuning decisions with other feedback control loops. This would require an overall model of the entire DBS, which quantitatively defines the system behaviour under all possible configurations. Due to its complexity, we consider it as not possible to create such a model of today's feature-overloaded DBMSs.

2.2 Towards Autonomic DBMS

In order to reduce the complexity of self-management, Weikum et al. advocate a radical departure towards a DBS that is based on RISC-style components [19]. Each component is supposed to have a limited functionality, which makes it possible to predict its behaviour for different configurations using mathematical models. To reduce the interaction complexity, the interfaces of the RISC components should be designed as narrow as possible. The proposed architecture uses rather coarse-grained components intended as building blocks for enterprise applications. In particular, the authors propose to create a select-project-join engine, which could either be used directly by an application, or upon which a multidimensional query engine could be plugged. To simplify self-configuration and -optimization, even SQL is proposed to be replaced by a narrow component API.

Instead of providing autonomic building blocks for enterprise applications, the focus of our research is the creation of an autonomic database. Still, the approach we follow for building an autonomic database is inspired by the RISC-style architecture proposal. As illustrated in Figure 3, we design a DBMS as a hierarchy of *autonomic elements*. In contrast to the proposal described in [19], these autonomic elements will be fine-grained, i.e. on system component level. For example, autonomic elements on the first hierarchy level could be chosen as the layers from Figure 2. Each layer could then be built using more specialized elements on the lower levels.

As shown in Figure 3, each autonomic element consists of two parts: an *autonomic element descriptor* and an *autonomic element implementation*. The descriptor defines the functional interface of the autonomic element, its configuration parameters, and the environmental resources upon which it depends. Most importantly, the descriptor contains a mathematical model, which quantitatively defines the behaviour of the autonomic element, depending on its configuration parameters, the available resources and its workload. Due to the limited functionality of the individual elements, we assume that a precise description of the element's behaviour will be possible. The second part of the autonomic element, the autonomic element implementation, is supposed to be the self-managing realization of the element's functional interface. To achieve this goal, every element must follow the feedback control loop design pattern. The realization of the functional implementation may again use other, sub-ordinate autonomic elements. In this way, the autonomic database can be composed as the hierarchy of autonomic elements shown in Figure 3.

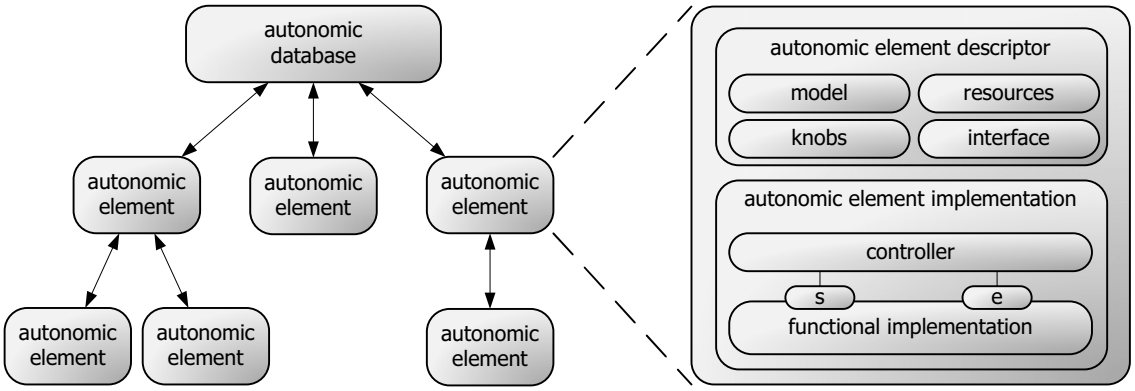

Figure 3: Autonomic DBMS as a Hierarchy of Autonomic Elements

Despite the fact that an autonomic element may be composed from sub-elements, its description must provide an overall model of its behaviour. Thus, for the element on the next-higher level in the system hierarchy the composition is opaque. Superordinate autonomic elements therefore only have to derive the optimal configuration of all of their *direct* subordinates, by using analytical or simulation techniques on the direct subordinates' mathematical models. Based on this result, every superordinate sets the *external configuration* of the subordinate elements, i.e. their tuning knobs and assigned resources. Every autonomic element is required to strictly adhere to this configuration. In particular, it must only use those resources (memory, CPU time, disk space, ...) that are assigned to it. This way, the consequences of resource sharing amongst the autonomic elements will be predictable and controllable throughout the entire system hierarchy. The external configuration defines the limits within which the feedback control loops of the subordinate elements may adapt their *internal configuration*. The internal configuration constitutes all configuration, tuning and maintenance decisions performed by the controller component of the feedback control loop. Autonomic elements which again use other autonomic elements, have to control the external configuration of the subordinates as part of their internal configuration.

With the described approach, the task of autonomic administration of the overall DBS is decentralized: every autonomic element manages only itself and configures its immediate sub-elements. From this point of view the top level node is considered as the autonomic database, which assigns the system resources to its sub-elements.

3. WORKLOAD MODELLING

The approach towards a fully autonomic database presented in Section 2 describes the long-term goal followed at our department. For our current work we focus on the tasks related to *global configuration and optimization* of a DBS. Thus, our results will be applicable to both current (layered or monolithic) DBMS architectures, and to the aspired system composed of hierarchically structured autonomic elements: For classic DBMSs, the global configuration and optimization reflects the self-management of the DBS from a bird's eye view, i.e. taking all system components into account. For the proposed hierarchically structured DBMS, it reflects the tasks of the control loop of the top-level node. In the following we first investigate the principles and challenges of global configuration and optimization of a DBS, while especially pointing out the workload-dependency. Afterwards, we describe our ideas about the usage of a workload model for this task.

3.1 Global Configuration and Optimization

Previous approaches to building a self-managing DBS have focused on specialized feedback control loops for certain aspects of DBS operation, e.g. for autonomic memory management. In the following, we will take a more general look at parameters influencing the self-management of a DBS.

From a bird's eye view, we can think of the entire DBS as a resource to be managed by a feedback control loop. This view is illustrated in Figure 4. In order to automatically derive the right configuration decisions, it is necessary to build a set of sensors, which supply the required monitoring information to the *self-management logic*. This logic continuously analyses the sensor data, and performs the reconfiguration via effectors. Using the feedback control loop metaphor for the task of DBS self-management makes our considerations independent of the underlying DBMS architecture. In particular, it provides a perspective that is general enough to ensure that deductions can be applied to both traditional (layered) DBMSs, and to the hierarchical, component-oriented approach described in Section 2.

From this perspective, there are two classes of monitoring information that have to be considered for self-management: database workload and database state. The *database workload* reflects the way the DBS is used in its particular environment. Typically, it is one of the main tasks of the DBA to adapt the out-of-the-box database product to its future usage. All physical design decisions for example heavily depend on the database workload. After the initial configuration has been set up, it is furthermore required to periodically check the DBS for a change in its usage. Thus, the database workload must be considered as an important input to the self-management decisions of an autonomic database controller. But workload monitoring only is not sufficient for self-management, because there are many maintenance tasks that are not workload-dependent. For example, the reorganisation of a table can greatly improve system performance, although there is no change in the DBS usage at all. Hence, the second input to the self-management logic is the information about the *state* of database objects and system components.

① workloads
② state of database objects.

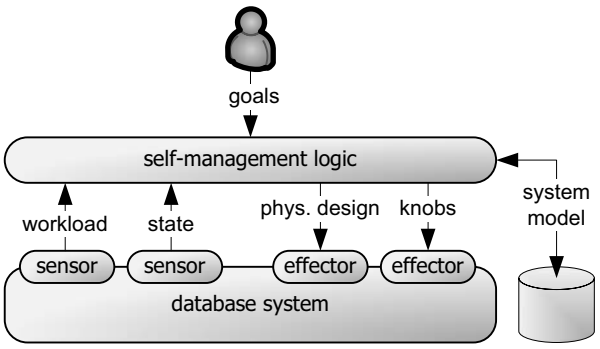

Figure 4: Feedback Control Loop Perspective on DBS Self-Management

In order to assess the monitored workload and state information, the controller must be able to predict the effects of possible re-configurations. For this purpose, it requires deep knowledge about the DBS internals in terms of a *system model*, that precisely describes the system components, their configuration parameters, and their performance. With the hierarchical approach for autonomic databases described in Section 2 we will be able to automatically derive this system model information from the autonomic elements' descriptors. For traditional, feature-rich DBMSs it will not be possible to create a complete quantitative system model. However, for these systems it is not the goal to make them fully autonomic, but making them more autonomic would be a great step ahead. Thus, even a crude model that approximates the behaviour and dependencies of the most important DBS components would be helpful.

Although the DBS is supposed to autonomically adapt to changes in its environment, it is essential for the customer to be able to set high-level *goals* for the self-management. Therefore [11] has defined that a DBS in an enterprise IT landscape in the "autonomic level" must be configurable via policies, which reflect service level agreements. For example, these policies could define whether the DBS should try to minimize resource usage, or maximize performance.

When the self-management logic has identified the need to perform a re-configuration of the DBS in order to meet the defined goals, it uses a set of effectors to actually put them into place. We distinguish two different kinds of effectors: Changes to the *physical design* include the management of access path structures, and all maintenance operations of the data like reorganization or compression. In contrast, the adaptation of tuning *knobs* includes all configurations of the DBMS-internal components.

3.2 Workload Shift Detection and Prediction

From the self-management parameters and tasks described in Section 3.1, the doctoral work outlined in this paper will focus on the monitoring and analysis of the database workload as a starting point. There are two reasons why we consider this topic as a promising research area: First, almost all self-management decisions depend on the workload because it provides information about the usage characteristics of a DBS. Even administrative tasks as for example the reorganisation of tables, which are solely triggered by state changes, should consider the workload to identify an appropriate maintenance period. Second, when comparing the different input parameters to the self-management logic (workload, state, goals, system model), the goals and system model change very rarely. In contrast, the database workload and the system state change much more frequently. So there is a strong need for an autonomic database to provide a reliable and light-weight analysis component for the latter two parameters. Of these, changes to the system state can only be detected in the individual system components. With current system architectures, we see no alternative to implementing highly specialized but independent feedback control loops. But with our long-term goal of a hierarchical composition of autonomic elements, the task of state monitoring is decentralized to the autonomic elements, where each element is able to self-monitor locally. In contrast to the state, the workload can be monitored in a centralized way at the DBS's top-level interface, i.e. at the SQL API. So independent of the underlying DBMS, it is possible to instrument standardized workload information.

Our approach towards efficient workload monitoring and analysis is based on the observation that the workload of an enterprise DBS is not composed randomly. Instead, the workload is determined by the applications that are using this DBS. Especially OLTP and reporting applications typically operate on a fixed set of *prepared statements*. They use a set of templates for SQL statements, which are just parameterised with the actual attribute values at runtime. Although for OLAP applications this is not the case (because the SQL statements depend on the user navigation through the data cube), it is likely that users will prefer certain evaluation directions. Hence, in an enterprise DBS scenario, the database workload typically is composed of a large set of fixed SQL statements that only differ in the actual attribute values, and a small set of statements that are user-generated. As long as the composition of this workload is fixed, there is no need for a DBMS to perform extensive analysis for re-configuration possibilities. We refer to this situation where the fluctuations in the workload are small and do not require self-management activities as a *stable workload*. For a DBS that is well-tuned for its current workload, it is important to know when there is a change in the workload. A significant change in the workload that could require re-configuration is called a *workload shift*.

Existing approaches to database workload analysis ([3], [6], [21]) are based on a heavy-weight analysis of an SQL trace in order to determine the appropriate physical design. Our idea for providing a continuous monitoring and analysis of the workload is to exploit the fact that this detailed analysis is required only when a workload shift occurs. Otherwise, it is sufficient to restrict the monitoring to the task of *workload shift detection*. Of course, this shift detection must be sufficiently robust against minor fluctuations in the workload. Examples of shifts that should be identified are the installation of a new version of a client application with a modified set of prepared SQL statements, or the release of a totally new client application. Also a change in the number of users of an application can impose a significant shift to the composition of the overall database workload. In addition to these *permanent shifts*, also *periodic shifts* must be considered: In a Data Warehousing application for example, the workload during daytime is typically made up of complex query statements, whereas at night it consists of mass updates. Or, as a longer-term periodic workload-shift, the end-of-month reporting of an OTLP application will gener-

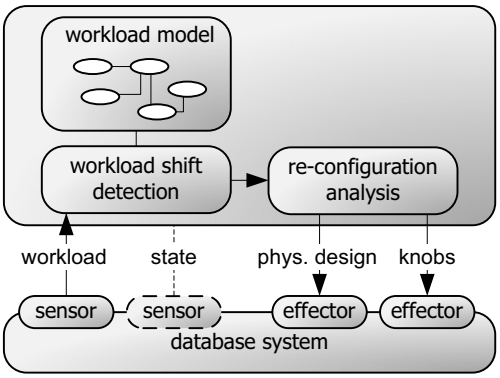

Figure 5: Two-staged workload analysis

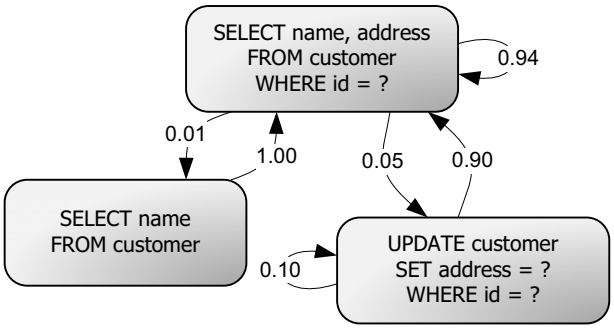

Figure 6: Workload Model using Markov Chain

ate a totally different workload than the usual operation.

In order to actually build this light-weight workload shift detection component for both permanent shift identification and periodic shifts prediction, we propose to maintain a *workload model* in the self-management logic. This model will permanently store an abstract representation of the DBS's workload. The self-management logic autonomically creates and maintains this model by monitoring the SQL API of the DBMS. After the model has been created in a learning phase, the current workload can be continuously compared to it to detect workload shifts. Only when a change is detected, the second, heavy-weight analysis stage is triggered, which assesses and executes possible re-configurations. The principle of the two-staged workload analysis employing a workload model is shown in Figure 5.

For the construction of a workload model there are contradicting requirements: On the one hand, the model must be as compact as possible to enable a light-weight comparison with the current load. On the other hand, the model should contain as much information as possible to be able to reflect long-term periodic workload shift patterns and to be robust against temporary fluctuations. It is the subject of our future research to identify the parameters that have to be reflected in the workload model. Furthermore, we need to find techniques for a compact representation of the SQL workload. It is for example not necessary to individually store every incarnation of a prepared statement that is processed by the DBMS. Instead the attribute values of statements could be ignored, thus keeping it in the model only once, while simply counting its executions. Besides the information about the statements themselves, it is also important to store temporal information in the model. Without this information it would not be possible to identify periodic workload shifts. Having settled the relevant parameters, a workload modelling technique has to be chosen. The selection process will consider aspects like the storage size, expression of workload patters, pattern matching support, and a strong theoretical background. Our current ideas for possible modelling languages include markov chains, petri nets, neuronal nets and fuzzy logic. Figure 6 exemplifies the use of markov chains of order 1 for the workload model. The states in this model represent the unattributed SQL statements of a workload, which are connected by their transition probabilities. It is important to note that Figure 6 is only an example to illustrate the idea of workload modelling: Although it provides a compact workload representation, it lacks information such as an execution count, temporal aspects, and transaction closures.

The workload model can be applied for three different use cases: *Workload shift identification* refers to the detection of permanent changes in the usage of the DBS. In this case, the self-management logic has to trigger the re-configuration of the DBS and to adapt the model to the new usage. In addition to the retrospective shift identification, the temporal information in the workload model can also be used for *workload shift prediction.* For example, an analysis of the model will enable the self-management logic to prepare the physical design to the conflicting requirements of the data warehousing scenario during day-time and night-time. As the patterns of periodic workload shifts must be reflected within the model, the model in this case must not be adapted. For the workload model will provide a compact representation of the DBS usage, it will serve as an efficient basis for general *DBS usage analysis* purposes. Hence, it can be used as substitute for the SQL trace in order to determine maintenance periods or to provide usage information to the DBA.

4. RELATED WORK

Prior work on the analysis of database workload can be classified into offline tools and online analysis. Whereas the online analysis techniques are just currently emerging, the offline tools have a longer history.

An early work that has identified the need for a deeper understanding of the database workload is REDWAR [20]. Designed as an offline-tool, it provides a characterization of the SQL trace, like structural information on the processed statements and runtime statistics. The REDWAR analysis results are presented as a report, which can be used to build a benchmark workload or to plan the physical design. Hence, REDWAR is neither built for continuous operation, nor can it recommend a physical design. Still, it has identified the criteria of SQL statements that affect DBS behaviour and performance.

Like REDWAR, the physical design advisors shipped by the database vendors ([3], [6], [21]) are offline tools. They recommend physical design structures for a given workload based on heavy-weight algorithms (e.g. knapsack) and expert knowledge. As the workload is considered as a set of statements, the advisors cannot exploit possible performance benefits from patterns in the workload. Also, they cannot compare their recommendations to previous results.

The only work so far which has identified the possible benefits from exploiting the order of statements in the workload

is [2]. In contrast to the advisors' set-based computations, this approach may adapt the physical design of the database for every statement (or set of statements). The authors describe algorithms that compute the physical designs which impose the least execution cost. These costs include the estimated costs for both statement processing and access path creation. The proposed approach suffers from the problem that the physical design can only be determined if the entire workload is known in advance. Even if it is assumed that the database workload is cyclic and consists of only one sequence that repeats continuously, a minor shift in the workload would make the analysis results unusable.

A promising work on online workload monitoring and analysis is COLT [17]. It continuously determines the possible benefit of all possible additional indexes by using the database optimizer in a "what-if-mode", i.e. the optimizer determines the statement execution costs assuming the indexes were present. To reduce its own overhead, COLT self-regulates the sampling rate with which it selects statements to be evaluated. The sampling rate is increased when a workload shift is assumed. In contrast to our approach, COLT does not maintain an explicit model of the workload. Instead, it stores a history of the processed statement to assess the current physical design. A shift is not identified by a significant deviation from a model, but from the estimated benefits from additional indexes. Without a workload model COLT is not able to predict periodic workload shifts, and it does not offer a compact analysis basis for other tools.

Like COLT, the online tuning approach [5] developed by Bruno and Chaudhuri continuously monitors the workload and adapts the physical design accordingly. This approach gathers information about the execution plan from the optimizer during query processing, and then uses a technique described in [4] to derive the execution costs for alternative physical designs from it. Thus, it reduces the overhead by not having to issue additional optimizer calls. Furthermore, it also takes into account index interactions and can avoid the problem of oscillation between physical designs. Compared to our approach, the online tuning proposed by Bruno and Chaudhuri is strictly focused on the efficient adaptation of the physical design. Other changes to the configuration of the DBS, which may also be caused by workload shifts, are not considered. And as it does not store a model of the workload, it is not able to identify patterns in the workload in order to predict workload shifts. Compared to the purely reactive approaches chosen in COLT and by Bruno and Chaudhuri, we expect that the resilience of noise will be higher when using an explicit workload model, because this will allow a detailed analysis of the workload history. However, both COLT and the online tuning approach will be a preferred partners for a future performance evaluations.

The Psychic-Sceptic Prediction Framework (PSP) [10] is a work by Elnaffar and Martin which has analysed the prediction of shifts from decision support (DSS) to OLTP workload. Analytical methods are offline applied to historical workload models in order to estimate the time interval for expected shifts. Only during this interval the framework actually performs an online-analysis of workload samples to detect the shift. The PSP does not build a model of the SQL workload, but maintains a list of "DSSness" indicators only. Thus, it is limited to the domain of OLTP to DSS shift detection. Later [15], the authors have identified the general need for both exploratory models (models of the monitored workload) and confirmatory models (models used by the self-management logic). However, only the need for these models is motivated, whereas a general approach for building them is not given.

5. FUTURE WORK

In Section 3 we have described our idea of creating and maintaining a workload model for the light-weight detection of workload shifts. A detailed investigation of this topic will be the first step in the research roadmap towards a self-managing DBS. In the following, we describe the different stages of this roadmap.

The starting point for our research is the *workload shift identification*. In this stage, we want to identify the parameters of the workload that are relevant for self-management decisions. The collected information must be sufficient for both optimization of physical design and setting of tuning knob values. A basic set of these parameters will be identified by a survey of the existing autonomic features of DBMSs. Based on these parameters, we will investigate the applicability of different modelling techniques like markov chains, multidimensional histograms or petri nets. To assess their individual advantages, we will realize prototypical implementations. Furthermore, we need to exactly define the notion of a significant shift in order to distinguish temporary fluctuations from permanent workload changes. Statistical testing methods might prove valuable for this purpose. At this stage, whenever a significant shift has been identified, triggering an external advisor tool is considered sufficient.

While in the first stage we focus on the identification of workload shifts, we aim at *autonomic re-configuration* in a second stage. Here, we plan to exploit the knowledge that can be gained by comparing an existing workload model against a "new" workload model that reflects workload changes. Furthermore, the workload model will be analysed to predict forthcoming periodic workload shifts. In order to translate this knowledge into appropriate self-management decisions, the development of a system model is essential. As building a full-fledged, quantitative system model for current DBMS architecture is considered impossible, we will start with a crude model of the most important components and their knobs. The results of the autonomic features survey conducted in the first stage will provide a good starting point for the selection of these knobs. With the proceeding development of the hierarchical, component-oriented approach for an autonomic database described in Section 2, the autonomic re-configuration logic will later on be able to exactly predict the effect of its decisions.

The final stage of the research roadmap is *goal-driven self-management*. In this stage, the administrator will have the possibility to control the autonomic database via high-level business goals. These business goals then must be translated into constraints for the processing of queries and manipulation operations in the DBMS. Thus, this stage demands a survey of the existing mechanisms for business goal definition. For the translation to the technical DBS goals, the workload model will be used as the analysis basis of the application's behaviour. During operation, the state of the DBS must be continuously compared against the derived DBS goals, and appropriate actions must be taken if the goals are not met. For this reason, the goal-driven self-management stage also requires mechanisms for the efficient monitoring and analysis of the DBS state.

6. CONCLUSIONS

This position paper has outlined the doctoral work in the area of autonomic database systems. The need for these systems is driven by the observation that today people cost are dominant in the total cost of ownership of DBSs. The current approaches for the implementation of autonomic databases fail to reach this goal: They are either too resource consuming for continuous operation, or they focus on a specific optimization problem without considering possible side-effects. As the key to overcoming these problems, we propose to build a feedback control loop for the overall DBS, which has system-wide control over all self-management activities. Its self-management logic must monitor the workload and state of the DBS, compare it against goals and derive the necessary re-configuration decisions using a system model. The challenge in building this logic is the minimization of the overhead. With the workload modelling described in Section 3 we have presented an approach that will greatly reduce this overhead for workload monitoring. In addition to permanent workload shift detection, it will also be used to predict shifts and to derive possible re-configuration actions using a system model. For it is considered impossible to define an overall quantitative system model for today's feature-overloaded DBMSs, we have illustrated our ideas about building a DBMS from hierarchically structured autonomic elements.

7. REFERENCES

[1] A. Aboulnaga, P. J. Haas, S. Lightstone, G. M. Lohman, V. Markl, I. Popivanov, and V. Raman. Automated statistics collection in DB2 UDB. In *Proceedings of the 30th International Conference on Very Large Data Bases*, pages 1146–1157. Morgan Kaufmann, 2004.

[2] S. Agrawal, E. Chu, and V. Narasayya. Automatic physical design tuning: workload as a sequence. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 683–694. ACM Press, 2006.

[3] S. Agrawal et al. Database Tuning Advisor for Microsoft SQL Server 2005. In *Proceedings of the 30th International Conference on Very Large Data Bases*, pages 1110–1121. Morgan Kaufmann, 2004.

[4] N. Bruno and S. Chaudhuri. To tune or not to tune?: a lightweight physical design alerter. In *Proceedings of the 32nd international conference on Very large data bases*, pages 499–510. VLDB Endowment, 2006.

[5] N. Bruno and S. Chaudhuri. An online approach to physical design tuning. In *Proceedings of the 23nd International Conference on Data Engineering*, pages 826–835. IEEE Computer Society, 2007.

[6] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait, and M. Ziauddin. Automatic SQL tuning in Oracle 10g. In *Proceedings of the 30th International Conference on Very Large Data Bases*, pages 1098–1109. Morgan Kaufmann, 2004.

[7] B. Dageville and K. Dias. Oracle's self-tuning architecture and solutions. *IEEE Data Engineering Bulletin*, 29(3), 2006.

[8] Y. Diao, J. L. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung. Self-managing systems: A control theory foundation. In *Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pages 441–448. IEEE Computer Society, 2005.

[9] S. Elnaffar, W. Powley, D. Benoit, and P. Martin. Today's DBMSs: How autonomic are they? In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, pages 651–655. IEEE Computer Society, 2003.

[10] S. S. Elnaffar and P. Martin. An intelligent framework for predicting shifts in the workloads of autonomic database management systems. In *Proceedings of IEEE International Conference on Advances in Intelligent Systems – Theory and Applications*, 2004.

[11] Ganik and Corbi. The dawning of the autonomic computing era. *IBM SYSTEMS JOURNAL*, 42(1):5–18, 2003.

[12] T. Härder and A. Reuter. Concepts for implementing a centralized database management system. In *Proceedings International Computing Symposium on Application Systems Development*, pages 28–59. B. G. Teubner, 1983.

[13] E. Kwan, S. Lightstone, A. Storm, and L. Wu. Automatic configuration for IBM DB2 Universal Database. Performance technical report, International Business Machines Corporation, 2002.

[14] T. Lahiri, A. Nithrkashyap, S. Kumar, B. Hirano, K. Pate, and P. Kumar. The self-managing database: Automatic SGA memory management. White paper, Oracle Corporation, 2003.

[15] P. Martin, S. Elnaffar, and T. Wasserman. Workload models for autonomic database management systems. In *Proceedings of the International Conference on Autonomic and Autonomous Systems*, page 10. IEEE Computer Society, 2006.

[16] S. Parekh et al. Throttling utilities in the IBM DB2 Universal Database server. In *Proceedings of the American Control Conference*, pages 1986–1991. IEEE Computer Society, 2004.

[17] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis. On-line index selection for shifting workloads. In *Proceedings of ICDE Workshops (SMDB 2007)*, pages 459–468. IEEE Computer Society, 2007.

[18] A. J. Storm, C. Garcia-Arellano, S. S. Lightstone, Y. Diao, and M. Surendra. Adaptive self-tuning memory in DB2. In *Proceedings of 32nd International Conference on Very Large Data Bases*, pages 1081–1092. ACM, 2006.

[19] G. Weikum, A. Mönkeberg, C. Hasse, and P. Zabback. Self-tuning database technology and information services: from wishful thinking to viable engineering. In *Proceedings of 28th International Conference on Very Large Data Bases*, pages 20–31. Morgan Kaufmann, 2002.

[20] P. Yu, M.-S. Chen, H.-U. Heiss, and S. Lee. On workload characterization of relational database environments. *IEEE Transactions on Software Engineering*, 18(4):347–355, 1992.

[21] D. C. Zilio et al. Recommending materialized views and indexes with IBM DB2 Design Advisor. In *Proceedings of the International Conference on Autonomic Computing*, pages 180–188. IEEE Computer Society, 2004.