Autonomic Databases: Detection of Workload Shifts with n-Gram-Models

Marc Holze and Norbert Ritter

University of Hamburg, Department of Informatics,
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
{holze,ritter}@informatik.uni-hamburg.de
http://www.informatik.uni-hamburg.de/

Abstract. Autonomic databases are intended to reduce the total cost of ownership for a database system by providing self-management functionality. The self-management decisions heavily depend on the database workload, as the workload influences both the physical design and the DBMS configuration. In particular, a database reconfiguration is required whenever there is a significant change, i.e. shift, in the workload.

In this paper we present an approach for continuous, light-weight workload monitoring in autonomic databases. Our concept is based on a workload model, which describes the typical workload of a particular DBS using n-Gram-Models. We show how this model can be used to detect significant workload changes. Additionally, a processing model for the instrumentation of the workload is proposed. We evaluate our approach using several workload shift scenarios.

1 Introduction

Vendors of commercial relational database management systems (DBMS) have continuously added new features to their products. While on the one hand this featurism makes DBMSs applicable for many different purposes, it on the other hand requires a subtle configuration for a particular environment. Commercial DBMS today offer hundreds of configuration parameters and physical design options that must be correctly chosen for a reasonable performance. Hence, it has been noticed ([1], [2]) that today the total cost of ownership for database systems (DBS) is not dominated by hardware- or software-costs anymore, but by "people cost" for database operation and maintenance. The ability of DBMS to perform self-management is considered as a way out of the high DBS maintenance costs. The DBMS is expected to autonomically perform maintenance operations and re-configurations when they are required, thus providing self-configuration, self-optimization, self-healing and self-protection [3].

The workload of a DBS has major influence on all DBMS self-optimization and self-configuration decisions, because it reflects the way the DBMS is used in its particular environment. For example, all physical design decisions heavily depend on the database workload, because the access paths should optimally support the typical query structure. Many configuration parameters also should be set according to the workload, e.g. the buffer- and sort-pool sizes. As the workload of a database typically evolves over time, it is required to regularly check whether the DBS is still optimally configured for the workload. Thus, in order to realize an autonomic database system, it is essential to perform continuous monitoring and analysis of the DBS workload.

Recently, commercial DBMS vendors have started to integrate *autonomic features* into their products, which help the DBA to adapt the DBS to the workload: *Advisors* are designed as offline tools that support DBAs with recommendations on specific administration tasks, e.g. physical design advisors ([1],[4],[5]). Although these advisors produce good results, their analysis is far too heavy-weight to be run continuously. In contrast, *Feedback Control Loops* constantly monitor and reconfigure a specific managed resource. To keep the monitoring and analysis overhead small, their view is limited to a specific domain, e.g. autonomic memory management ([6], [7]). For this reason, feedback control loops focus on the workload of only the component that they manage (e.g. buffer pools) and cannot determine how their observations relate to the workload in other components or the entire DBS.

The work presented in this paper focuses on the continuous, light-weight monitoring and analysis of the overall database workload. For this purpose, we follow the idea of a two-staged workload analysis that we have described in [8]. This idea is based on the observation that in real-world DBS the workload is determined by the applications using the database. Typically, these applications work with a fixed set of static SQL statements or statement-templates, leading to specific usage patterns. In times where the workload is stable, i.e. there are only minor fluctuations in the way the database is used, there is no need to perform extensive reconfiguration analysis. Our idea of two-staged workload-analysis (see Fig. 1), exploits this observation: In the first stage, the self-management logic continuously monitors information about the current workload of the database. This stage only performs a light-weight comparison of the observed workload against a workload model, which describes the typical usage of the DBS. Only if a significant change (*workload shift*) from the typical usage is detected, the second, heavy-weight reconfiguration analysis stage is started.

The contribution of this paper is an approach based on n-Gram-Models for the detection of workload shifts in the first stage of workload analysis (the second stage is not in the focus of this paper). We illustrate how to automatically learn a workload model, how to assess an observed workload based on a n-Gram-Model, and how to adapt the model to a changed DBS usage. For this purpose we give an algorithm for the management of the workload model lifecycle. Furthermore, we propose a flexible approach for the instrumentation of the database workload.

The paper is organized as follows: The problem description in Section 2 discusses the requirements of a workload model. Section 3 then first describes a flexible approach for monitoring database workload, before Section 4 presents the contents and management of the workload model for workload shift detection. We evaluate our approach in Section 5 and discuss related work in Section 6. We conclude with a summary and outlook on future work in Section 7.

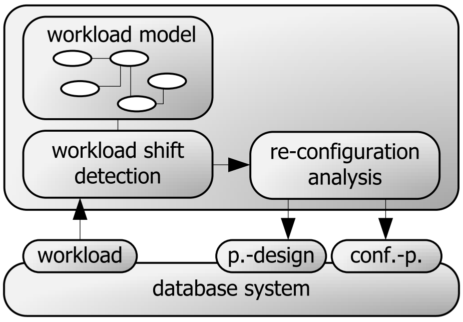

Fig. 1. Two-Staged Workload Analysis

2 Problem Description

The purpose of workload shift detection is the identification of situations when the usage starts to significantly change. The usage changes that must be detectable are all those scenarios, which usually require a DBA to re-analyse the configuration of a DBS. In particular, we consider the following scenarios as relevant shift situations:

- *New Applications:* When new database applications are deployed, they will cause additional workload on the DBS. To meet the performance requirements of all applications it may be necessary to perform DBS reconfigurations.
- *Obsolete Applications:* Workload that does no longer occur may also require DBS reconfigurations. If for example one database application is undeployed, some physical access structures or bufferpools may become obsolete.
- *Modified Applications:* With the installation of new releases, the functionality of enterprise applications usually evolves over time. The resulting workload may change significantly in every release.
- *Application Usage Changes:* Even with a fixed number and type of database applications, the DBS workload may change due to the user behaviour. If for example a reporting tool is used by twice the number of end users, the composition of the overall DBS workload may shift significantly.

In the above scenarios, the change in the workload is *permanent*, i.e. the changes will persist in the future. In addition to permanent shifts, databases often also face periodic workload shifts. Periodic shifts occur when a certain type of database usage occurs in regular intervals. For the workload shift detection, it is important to distinguish between two types of periodic shifts:

- *Short-Term Patterns:* These patterns refer to periodic workload shifts where the interval between the usage changes is small, e.g. batch updates in an 15 minutes interval or hourly reports. In this case, the periodic reconfiguration analysis and execution effort is likely to exceed the resulting benefit. Their actual length is installation-specific and must be configurable. Short-term patterns must *not* be detected as a workload shift.

– *Long-Term Patterns:* In cases of long-term patterns the period length justifies the reconfiguration analysis effort. An example is a DWH-scenario, where the DBS is loaded at night, and queried at day-time. These two distinct workload types require completely different DBS configurations and should therefore be detected as workload shifts. The workload model should furthermore provide the possibility to predict these periodic patterns, as this would make it possible to perform the reconfigurations proactively.

The workload shift detection must detection the *detection of the usage change scenarios* described above. This requirement implies that the detection mechanism may identify all usage changes which *might* require a DBS reconfiguration. The decision whether or not this usage change actually leads to a reconfiguration does not have to be covered by the shift detection. To provide a reliable reconfiguration indicator, the workload shift detection must furthermore ensure that *only* the described usage changes are detected. Minor workload fluctuations must be identifiable as such (*resilience to noise*).

The goal of continuous workload monitoring is the autonomic reconfiguration of the DBS, in order to lower the total cost of ownership for a DBS. Hence, the workload model must be *learnable and adaptable* by the DBMS without causing any effort for the DBA. For a light-weight shift detection, the learned workload must be represented in a *compact model*, i.e. the model should abstract from unnecessary details. One of the lessons learned from the first autonomic features in commercial DBMSs was that users do not trust in autonomic functionality [9]. For this reason the workload model should be *comprehensible*.

From these workload shift detection requirements, the following key challenges can be identified: In order to realize a DBS usage change detection, the ability for *workload instrumentation* is an essential prerequisite. It has to be decided

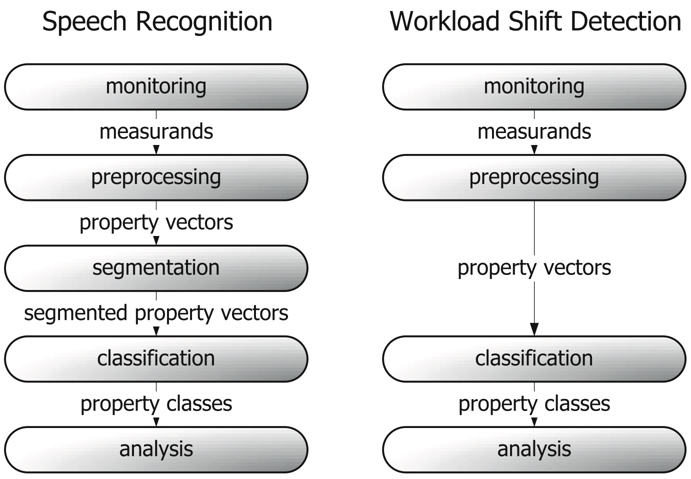

Fig. 2. Processing Models in Speech Recognition and DBS Workload Analysis

which metrics of the operating system or DBMS are monitored for this purpose, while imposing as little overhead as possible. The measured workload information must be represented in a compact workload model, i.e. the information in the model must abstract from minor differences in the measurements. Hence, the workload shift detection must perform a *workload classification* of similar workload situations. As the model must be automatically learned, an appropriate modelling technique has to be chosen that supports *model learning*. Afterwards, the workload shift detection has to perform *workload assessment*, i.e. the DBS workload must be continuously compared against the learned model. Metrics must be found that can be used to describe how well the observed workload matches the model. These metrics have to be assessed in a way that on the one hand reliably detects the usage change scenarios, and on the other hand assures resilience to noise. Whenever a significant workload shift has been detected, an alarm should be raised which triggers the DBS reconfiguration. In this case, a *model adaptation* must be performed automatically in order to represent the changed workload in the model.

3 Processing Model

An analysis of the key challenges described in Section 2 has shown that they are similar to the challenges in pattern recognition, especially speech recognition. This field of computer science also requires the assessment of measured data with the help of (language) models. As a well-established and tested processing model has already been developed in this discipline, we adapted this model for the purpose of workload shift detection. Fig. 2 provides an overview of the pattern recognition processing model as described in [10] and its adaptations for DBS workload analysis. The remainder of this section describes those processing stages of the processing model in detail.

The first stage in the processing model is *monitoring*. In principle, there are many possibilities for the monitoring of DBS workload. Some existing autonomic features like the IBM DB2 Utility Throttling [11] use the CPU usage, memory usage or I/O activity operating system metrics. Others monitor database-internal metrics of specific sub-components, like the buffer pool hit ratios or optimizer cost estimations. In contrast to these approaches, we consider monitoring the DBS workload at the SQL API (e.g. via a proxy or SQL statement monitor) as appropriate for workload shift detection. The reason is that the load of the DBMS-components directly depends on the set of observed SQL statements. Compared to monitoring the workload at all DBMS components independently, this centralized monitoring will impose less overhead. Furthermore, monitoring this standardized API makes the approach applicable for all relational DBS.

In the pattern recognition processing model, the monitored information (measurands) is preprocessed, e.g. filtered and described in terms of multi-dimensional property vectors. *Preprocessing* the monitored DBS workload is also reasonable for workload shift detection in order to filter certain types of SQL which should not be considered in the model, e.g. DDL statements. The conversion of the monitored workload into multi-dimensional property vectors is future work.

On the stream of preprocessed property vectors the speech recognition processing model next performs *segmentation* into meaningful sections, i.e. splitting at the word boundaries. As the DBS workload is already segmented into the individual statements, this stage of the processing model may be skipped.

Classification refers to placing individual observations into groups according to certain characteristics. For speech, this stage maps the property vector segments to a symbolic representation of the spoken words. Only some characteristics of the recorded speech are evaluated for this purpose (thus allowing the same word to be pronounced/spoken in different ways). Classification is also required for DBS workload in order to fulfil the requirement of a compact workload model. Otherwise every monitored SQL statement with a distinct statement text would result in a separate class in the model. For a database with one single table of 100,000 customers that are accessed by their ID, the workload model would comprise 100,000 different SQL statements. From a DBS-load perspective, all statements that cause a similar load on the DBS should be classified into common classes, e.g. by comparing access plans (*DBS-load classification*). But the workload-shift-scenarios that we defined in Section 2 suggest a more semantic view on the classification. From this perspective, similar actions in the application programs should be classified into common classes (*semantic classification*). Both classification approaches have major drawbacks: The semantic classification would require the manual definition and maintenance of classification rules. This manual effort thwarts the goals of autonomic databases. The DBS-load classification would lead to considerable effort for the comparison of access plans, and it would not be easily comprehensible. Hence, we have chosen a simple *statement type classification* mechanism that is inexpensive and can be performed without DBA configuration effort: The SQL statements are classified by replacing all concrete parameter values in the statement text with a wildcard-character. If a class that represents the resulting un-parametrized string already exist, then the SQL statement is assigned to this class. Otherwise, a new class is dynamically created. The statement type classification introduces inaccuracy into the model by not considering effort differences due to data skew or locality of transactions. Also the number of classes cannot be known or limited in advance, and identical queries written in syntactically different ways will be assigned to different classes. Still, this straight-forward classification has produced good results (see Section 5) while imposing little overhead. More sophisticated classification algorithms can be evaluated and – due to the strict separation of processing stages – easily integrated in the future.

The last stage of the processing model is the *analysis* of the classified information. In the context of speech recognition this typically is language understanding. For workload shift detection, it comprises the comparison of the classified DBS workload with the workload model. The workload modelling technique and model management concepts that have been developed for this purpose are described in the following Section 4.

4 Workload Analysis

After the monitored information has been classified according to Section 3, it is compared to a workload model in order to detect workload shifts. This section first presents the modelling technique chosen for the representation of DBS workload and afterwards describes the management of the workload model.

4.1 DBS Workload Modelling

Our approach for modelling the DBS workload is based on the idea of creating a model of the typical database application *behaviour*. This behaviour describes the interaction of the applications with the database, i.e. the SQL statements they submit and the context in which these appear. From another perspective, this can be seen as modelling the "language" the applications speak to the DBS.

Considering the goal of workload shift detection, it is not required to recognize a single new statement in the workload. Instead, significant changes in the overall composition of statements must efficiently be recognized. So statistical methods are applicable for this task. In our concept we have decided to use *n-Gram-Models*, which have already been successfully applied in other problem domains (e.g. bioinformatics and natural language processing). These models are learnable, offer a strong formal background with existing algorithms, and can be represented in a comprehensible, graphical way. n-Gram-Models are based on markov chains, which model the behaviour of an event source behaving according to a stochastic process. This stochastic process generates a sequence of events $\mathbf{X} = (X_1, X_2, ..., X_t : t \in T)$, where each event X_t takes a value of $S = (s_1, s_2, ..., s_k : k \in \mathbb{N})$. To be a markov chain, the probability of events must adhere to the markov property, i.e. it must not depend on the entire history of previous events, but only on a limited history of length m (*order*). From DBS perspective, database applications can be considered as a stochastic process generating a workload $\mathbf{W} = (W_1, W_2, ..., W_t)$ as a sequence of SQL statements. To represent this workload in a markov chain model, each statement W_t must be mapped to an event X_t taking the value s_i. State s_i can be seen as the internal state of the application when generating the SQL statement at time t. We have investigated two alternative modelling approaches for this purpose:

- *Inference of Transactions:* All statements issued in the same transaction type are mapped to one model state. This approach results in a compact workload model with a small number of states. But as there usually is no information about the type of transaction in the workload $\mathbf{W}$, it is necessary to infer the transaction types from the observed statements. This is a complex task for procedurally controlled transactions [12], where the number and type of SQL statements in a transaction varies depending on loop/branch-conditions.
- *Statement Types:* In this approach we directly map each statement type to a markov model state. The transition probabilities are computed according to the ordering in $\mathbf{W}$. On the one hand this approach results in model with more states, while on the other hand we avoid the problem of transaction inference.

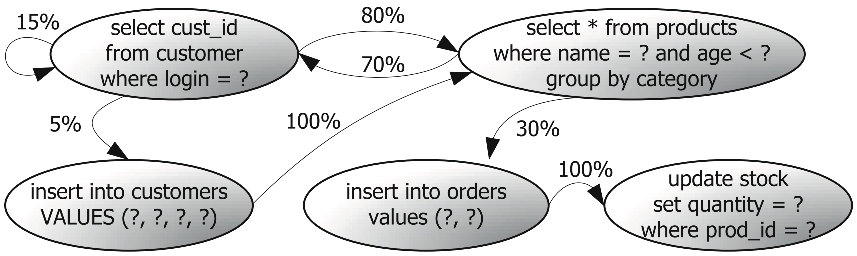
15%
select cust_id from customer where login = ?
80%
70%
select * from products where name = ? and age < ? group by category
5%
100%
30%
insert into customers VALUES (?, ?, ?, ?)
insert into orders values (?, ?)
100%
update stock set quantity = ? where prod_id = ?

Fig. 3. Example for DBS-Workload modelled as Markov Chain of Order 1

For our concept of workload shift detection we have chosen the statement type mapping, because it produces less overhead than the inference of transactions (cf. [12]). In the processing model presented in Section 3 we have furthermore proposed a classification approach that already performs this task. In the following we therefore assume that the workload $\mathbf{W}$ already is given in terms of statement types. So $\mathbf{W}$ is equal to $\mathbf{X}$ in our case and takes the values of the model states S. Fig. 3 shows an illustration of a markov-chain workload model using the statement type mapping. Each node of this graph represents a statement type. The probability of forthcoming events in this example depends only on the current state (markov chain of order 1). These *transitions* between states are illustrated as attributed edges.

As described above, the prerequisite for creating a markov-chain model is that the event source satisfies the markov property. For this reason, creating a markov-chain model requires thorough knowledge about the stochastic process and a manual modelling of its behaviour. In order to avoid this requirement, n-Gram-Models consider the markov property only as an approximation. Hence, the order of the markov chains determines how well the model approximates the underlying process. With this approximation, n-Gram-Models can automatically be learned from training data by counting the occurrences of events:

$$P(W_{t_i} \mid W_{t_{i-n}}, ..., W_{t_{i-1}}) = \frac{count(W_{t_{i-n}}, ..., W_{t_{i-1}}, W_{t_i})}{count(W_{t_{i-n}}, ..., W_{t_{i-1}})} \; . \tag{1}$$

So for the creation and analysis of n-Gram-Models, only sub-sequences with the length n of the events $\mathbf{W}$ are considered (n-Gram-Models result in markov-chains of order $n-1$).

As n-Gram-Models are only an approximation of the real world process, all analysis on the model faces the problem of *unseen events*. Unseen events are events that have not been observed, either because the event W_t has not occurred yet at all, or because it has not been seen with a particular history $(W_{t-n}, ..., W_{t-1})$ yet. To prevent these unseen events from being evaluated to a probability of 0, there are a number of techniques in literature ([13], [14]). For our evaluation described in Section 5) we have chosen a combination of *absolute discounting* and *backing off* for this purpose.

4.2 Workload Shift Detection

Having introduced the concepts of DBS workload modelling, we will now describe how workload shift detection is realized on this basis. For this purpose it is essential to assess how well the observed DBS workload is described by the workload model. In statistics, the metric commonly used to assess the quality of a probability model is the *perplexity*. Informally, the perplexity describes how "surprised" the model is by the observed workload. For an observed workload $\mathbf{W}$, the perplexity PP of an DBS workload n-Gram-Model is computed as

$$PP(\mathbf{W}) = \left(\prod_{t=1}^{T} P(W_t \mid W_{t-n} ... W_{t-1}) \right)^{-\frac{1}{T}} . \tag{2}$$

For the computation of the perplexity, the probabilities $P(W_t)$ of the individual SQL statements are taken from the workload model. Using the perplexity PP as a metric, the decision of whether or not the model sufficiently well describes the actual DBS workload results in choosing a threshold for the perplexity. The appropriate value for the perplexity threshold depends on the installation-specific workload characteristics. From Equation 2 it is obvious that the expected perplexity for a completely random workload is equal to the number of statement classes. Procedurally controlled workloads result in lower perplexities. In our evaluation (Section 5) a perplexity value of $1,5 * |S|$ has produced good results for a mixed workload. It is important to note that the perplexity definition only implicitly considers obsolete workload: A workload model will determine low perplexity values for a workload even if there are many obsolete states in the model.

In order to realize workload shift detection for an autonomic database, the workload model must be maintained without any user interaction. Fig. 4 illustrates the lifecycle that is used to automatically manage the workload model in our approach. As the goal of autonomic databases is the reduction of the maintenance costs, creating an initial workload model from a set of manually provided training data is not suitable. Instead, the workload model must be automatically learned. The shift detection logic therefore always creates a new workload model in state "learning". This state indicates that the model must still be trained. After the model has learned the typical DBS workload, it is switched to state "stable". It remains in this state as long as the fluctuations in the DBS workload are small. When a significant change is detected the workload model switches to state "adapting", where it must again be trained for the changed workload before returning to state "stable" again.

Considering the power of the selected n-Gram-Modelling approach and the above lifecycle, there remain two main challenges that must be addressed in order to realize workload shift detection: The decision on the end of the learning/adapting phase and the detection of outdated states and transitions in the model.

The decision on the end of the "learning" and "adapting" phases for a workload model can be made by monitoring the perplexity values. If the perplexity has not exceeded the perplexity threshold for a certain period of time, it can be assumed that the model is stable. The length of this period is determined

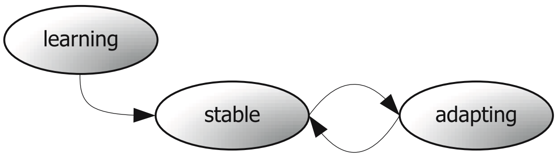

Fig. 4. Workload Model Lifecycle for DBS Workload Shift Detection

by the configurable length of short-term patterns (cf. Section 2). As according to our requirement definition these patterns should not be detected as workload shifts, they must be represented in the model. Requiring the perplexity to be strictly below the threshold for the entire pattern length may lead to long learning intervals, because a single probe that exceeded this threshold would restart the counter. Thus, we consider it sufficient if a certain percentage (e.g. 80%) of perplexity values in the past short-term pattern period length has had values below the threshold (stability factor).

Whereas the perplexity metric provides a good indicator for new events or a different event composition, it is not meaningful in case of obsolete events. Hence, we have included the following ageing mechanism in the workload model management: Every transition in the model is attributed with a timestamp, which indicates the most recent observation of the transition. When the age of a transition has exceeded a certain age, their information is removed from the model. For a consistent assessment of deviations from the workload changes, we again utilizes the perplexity metric. We generate an artificial sample probe from the workload model before the removal of model elements, and then compute the perplexity for this sample probe with the new, adapted model. If the perplexity value exceeds the threshold, a workload shift due to obsolete workload has occurred.

The overall workload shift detection logic is illustrated as pseudo code in Algorithm 4.1. In a first step, the shift detection logic converts the observed workload **W** to n-Grams (**1**). Every n-Gram consists of the observed event (statement type) and the history of $n-1$ preceding events, where $n-1$ is given as a predefined value CHAIN_LENGTH[1]. Based on these n-Grams, it computes the perplexity for the current model according to Equation 2 (**2**). The subsequent actions depend on the state of the model: If the model is in state “learning” or “adapting”, then the transition probabilities in the model are recalculated according to Equation 1 for all probes that exceed the perplexity threshold THRESH (**17**). In addition, new states are added to the model for previously unknown statements in this step. Afterwards the workload shift detection logic checks whether the required percentage STAB of probes in the past short-term pattern period (of length SHORT) has taken perplexity values below the threshold (**18**). If this is the case, then the model is switched to state “stable” (**21**). A workload shift alarm is raised (**19-20**) now that the workload has reached a stability level again.

[1] Typical values for the chain lengths in n-Gram-Models are 2 and 3.

Algorithm 4.1. Model Management for Workload Shift Detection

```
Algorithm: detectShift
Input: an observed workload W, a workload model model
1  nGrams ← convertToNGrams(W, $CHAIN\_LENGTH$);
2  pp ← model.computePerplexity(nGrams);
3  hist ← hist.add($pp > THRESH$);
4  if $model.state = $"stable" then
5      if $pp > THRESH$ then
6          if $hist.cntStable(SHORT) < (STAB * hist.countAll(SHORT))$ then
7              model.reset();
8              model.state ← "adapting";
9      else
10         if $model.hasOutdatedStatesAndTransitions(2 * SHORT)$ then
11             probeNGrams ← model.generateProbe();
12             model.removeOutdatedStatesAndTransitions(SHORT);
13             if $model.computePerplexity(probeNGrams) > THRES$ then
14                 raiseAlert("DBS Workload Shift. Reconfiguration required.");
15 if model.state = "learning" or model.state = "adapting" then
16     if $pp > THRESH$ then
17         model.learnFrom(nGrams);
18     else if $hist.cntStable(SHORT) > (STAB * hist.cntAll(SHORT))$ then
19         if model.state = "adapting" then
20             raiseAlert("DBS Workload Shift. Reconfiguration required");
21         model.state ← "stable";
22     if model.state = "adapting" then
23         model.removeOutdatedStatesAndTransitions(SHORT);
```

In contrast, while the model is in state "stable", the model is not adapted at all. That is, even if there are differences between the observed workload and the model, the transition probabilities and states are not updated. The workload shift detection logic solely checks whether the percentage STAB of perplexity values has exceeded the threshold, and changes the model state to "adapting" in this case (**5-8**). Afterwards, the ageing mechanism described above is applied to the model (**10-14**).

5 Evaluation

For the evaluation of our concepts we have created the testbed illustrated in Fig. 5. Two sample applications (DVD-Shop and Reporting) are used to generate workload on a DBS. The DVD-Shop is a simple web shop based on a sample application by Dell Labs [15], whereas the Reporting generates custom reports from the shop data. As a LoadGenerator for the applications we have used Apache JMeter. The workload information is polled from the DBS by a prototypical implementation of the workload shift detection (WSD Prototype) in regular intervals. In this prototype we have realized the processing model as described in Section 3. All source-specific preprocessing steps (i.e. monitoring and filtering)

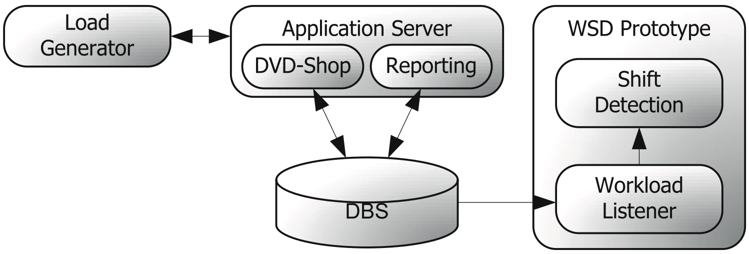

Fig. 5. Testbed for Evaluation of DBS Workload Shift Detection Prototype

of the workload information are encapsulated in a WorkloadListener component. The observed workload is then passed to the ShiftDetection component, where it is classified (according to the statement types) and analysed.

In order to achieve an efficient implementation of the n-Gram-Model in the ShiftDetection component, we have followed the approach proposed in [10]: To store the transition probabilities, we use a suffix tree which only stores the events and histories actually seen. Compared to using a transition matrix, this approach leads to a very compact model. To prevent the probability from taking a value of 0 for previously unseen events we have implemented a combination of the techniques *absolute discounting* and *backing off* [13].

The evaluation of our workload shift detection approach is based on the requirements defined in Section 2. Hence we have first defined the workload listed in Table 1a. The workloads Shop1 and Shop2 represent different user behaviours on the web shop application, whereas Rep1 and Rep2 simulate different releases of the reporting application (where the second release has an increased functionality, i.e. it offers twice the number of reports). From these workloads we have then composed the test cases TC1-TC9, which are described in Table 1b. The test cases cover the shift scenarios as defined in Section 2 (New Applications, Obsolete Applications, Modified Applications, Application Usage Changes, Long-Term-Patterns) and the requirements of resilience to noise and automatic learning/adapting.

We have executed the test cases TC1-TC9 in the testbed described above. For each of the test cases, the perplexity-values are plotted in Fig. 6. These values have been computed with a probe size of 100 statements on a 3-Gram-Model. In our tests, the maximum length for short-term patterns SHORT was defined as 180 seconds, the stability factor STAB as 0.8. The perplexity threshold THRESH was set to 30, which was 50% above the number of statement types in the test. For each test case Fig. 6 also denotes the changes in the model state for this parametrization (S denotes a change to state “stable”, A a change to state “adapting”, D for a deletion of outdated model states).

The results in Fig. 6 prove that the model is reliably learned from the observed workload (TC1), and that it automatically adapts quickly in case of usage changes (TC3). Minor fluctuations or short-term patterns in the workload (caused by the probabilistic workload generation by JMeter and explicitly

Table 1. Definition of Test Scenarios

(a) Workloads

Workl.		Action	Rep.	TT	P.
Shop1	1.	Login	1	5-10s	1.0
	2.	Search	1	0s	1.0
	3.	AddToCart	1	0s	1.0
	4.	Checkout	1	0s	1.0
	5.	Confirm	1	0s	1.0
Shop2	1.	Login	1	5-10s	0.5
		Register	1	0-5s	0.5
	2.	Search	10-20	0s	1.0
	3.	AddToCart	1	0s	1.0
	4.	Checkout	1	0s	1.0
	5.	Confirm	1	0s	1.0
Rep1	1.	Report 1	1	0-5s	0.2
		⋮	⋮	⋮	⋮
		Report 5	1	0-5s	0.2
Rep2	1.	Report 1	1	0-5s	0.1
		⋮	⋮	⋮	⋮
		Report 10	1	0-5s	0.1

(b) Test Cases

Scenario	Workl.	Users	Interv.
TC1: Learning	Shop1	10	[0;400]
TC2: R. to Noise	Shop1	10	[0;400]
	Shop2	1	[200;300]
TC3: Adaptation	Shop1	10	[0;200]
	Shop2	10	[200;400]
TC4: New App.	Shop1	10	[0;400]
	Rep1	10	[200;400]
TC5: Obsol. App.	Shop1	10	[0;800]
	Rep1	10	[0;200]
TC6: Mod. App.	Rep1	10	[0;200]
	Rep2	10	[200;400]
TC7: Usage Ch.	Shop1	10	[0;400]
	Rep1	2	[0;200]
	Rep1	10	[200;400]
TC8: Long Patt.	Shop1	10	[0;400]
	Rep1	10	[400;800]
	Shop1	10	[800;1200]
	Rep1	10	[1200;1600]
TC9: Short Patt.	Shop1	10	[0;100]
	Rep1	10	[100;200]
	Shop1	10	[200;300]
	⋮	⋮	⋮
	Rep1	10	[1500;1600]

induced in TC3 and TC9), do not cause a model instability. In contrast, all shift scenarios (TC4, TC5, TC6, TC7, TC8) are reliably detected as workload shifts.

The computation times for the perplexity of a probe and for the adaptation of a probe depend on the probe size, the number of statement classes and the markov chain length. Fig. 7 illustrates the computation times on a desktop workstation (clock cycle: 3 GHz/ memory:1 GB) for different chain lengths m. As the probe size can be considered as a scaling constant (cf. Equation 2), it has been set to a fixed value of 100. The results show that the proposed workload shift detection is very efficient while the model is stable, because even with a long chain length $m = 10$ the computation of the perplexity is always less than $50ms$. The adaptation of a model in the "learning" or "adapting" states requires more overhead, but these adaptations will be necessary only very rarely.

6 Related Work

All major commercial DBMSs today offer advisors, which recommend a good physical design for a given workload ([1], [4], [5]). Due to their heavy-weight analysis algorithms (e.g. knapsack) they are designed as offline-tools, and must be

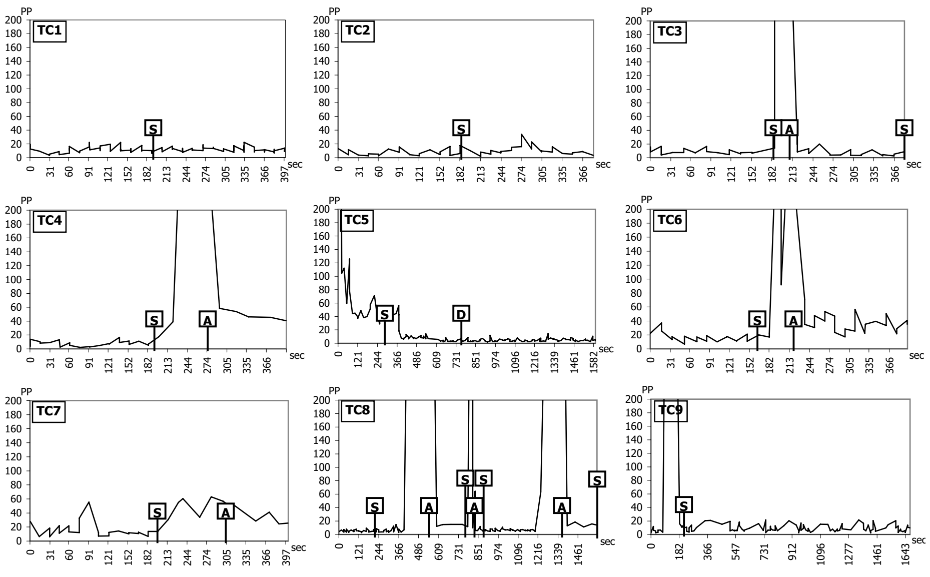

Fig. 6. Test Results for Test Cases TC1-TC9

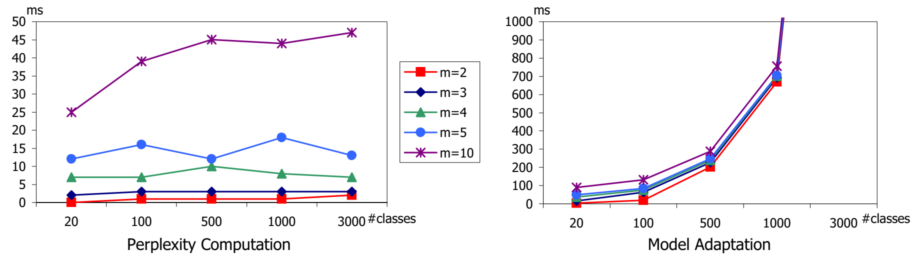

Fig. 7. Perplexity Computation Times for different Model Configurations

explicitly started by the DBA when he suspects possible improvements. Hence, these advisors are intended for the reconfiguration analysis rather than workload shift detection.

In contrast to the advisors, COLT [16] performs online workload monitoring and analysis. By using the database optimizer in a "what-if-mode", COLT continuously determines the possible benefit of hypothetical additional indexes. In contrast to our approach, COLT focuses on the online adaptation of the physical design. It does not provide a general-purpose workload shift detection, but is intended as an online-tool for autonomic index management.

The online tuning approach [17] by Bruno and Chaudhuri gathers execution plan information from the optimizer during normal query processing. From this information, it can directly derive the execution costs for alternative physical designs without having to issue additional calls to the optimizer. So like COLT, the approach in [17] focuses on the efficient online-adaptation of the physical design. Other changes to the configuration of the DBS, which may also be caused by workload shifts, are not considered.

Elnaffar et al. [18] propose the use of a workload model for the automatic classification of database workload in order to distinguish between OLTP and DSS workload. However, the workload model in their approach does not actually represent the database workload. It is instead a decision tree that is used for classification. Hence, their approach is limited on the detection of shifts between OLTP and DSS workloads. In a later work [19], the authors have also identified the general need for exploratory models (models of the monitored workload), but an approach for building them is not given.

N-Gram-Models have also been used in [12] for creating a statistical model of database workload. This work is directed at the inference of user sessions from SQL statement traces. Heavy-weight analysis algorithms are applied on these models in order to identify and cluster procedurally controlled sessions. The information about the user sessions is intended to be used for the prediction of queries based on the queries already submitted, e.g. to optimize the cache replacement strategies.

7 Conclusion

The workload has major influence on the configuration and optimization of a DBS. Hence, for a self-managing database systems, it is an essential prerequisite to continuously monitor and analyse the database workload. In this paper we have presented an approach based on n-Gram-Models, which detects shifts in the typical workload of a database. Its results provide a strong indicator for when a detailed reconfiguration analysis should be performed. Our evaluation has shown that the approach is very light-weight and can be used to perform continuous monitoring. In addition, it reliably detects significant changes and is resilient to noise.

In the future, we are going to extend our concept in two directions: On the one hand, long-term patterns should not only be detected as workload shifts, but should be reflected in the workload model. This extension will support the prediction of periodic changes in the workload and the proactive adaptation of the DBSs configuration. On the other hand, we are working on a more sophisticated classification algorithm than statement type identification. Our goal is a classification that allows the definition of a fixed (upper) limit of classes, thus providing an upper bound for the workload shift detection effort.

References

1. Agrawal, S., et al.: Database tuning advisor for Microsoft SQL Server 2005. In: Proceedings of the 30th International Conference on Very Large Data Bases, pp. 1110–1121. Morgan Kaufmann, San Francisco (2004)
2. Dageville, B., Dias, K.: Oracle's self-tuning architecture and solutions. IEEE Data Engineering Bulletin 29(3) (2006)

3. Ganik,, Corbi,: The dawning of the autonomic computing era. Ibm Systems Journal 42(1), 5–18 (2003)
4. Dageville, B., et al.: Automatic SQL tuning in Oracle 10g. In: Proceedings of the 30th International Conference on Very Large Data Bases, pp. 1098–1109. Morgan Kaufmann, San Francisco (2004)
5. Zilio, D.C., et al.: Recommending materialized views and indexes with IBM DB2 design advisor. In: Proceedings of the International Conference on Autonomic Computing, pp. 180–188. IEEE Computer Society, Los Alamitos (2004)
6. Lahiri, T., et al.: The self-managing database: Automatic SGA memory management. White paper, Oracle Corporation (2003)
7. Storm, A.J., et al.: Adaptive self-tuning memory in DB2. In: Proceedings of the 32nd International Conference on Very Large Data Bases, pp. 1081–1092. ACM, New York (2006)
8. Holze, M., Ritter, N.: Towards workload shift detection and prediction for autonomic databases. In: Proceedings of the ACM first Ph.D. workshop in CIKM, pp. 109–116. ACM Press, New York (2007)
9. Lightstone, S., et al.: Making DB2 products self-managing: Strategies and experiences. IEEE Data Engineering Bulletin 29(3) (2006)
10. Fink, G.: Markov Models for Pattern Recognition, 1st edn. Springer, Heidelberg (2008)
11. Parekh, S., et al.: Throttling utilities in the IBM DB2 universal database server. In: Proceedings of the American Control Conference, pp. 1986–1991. IEEE Computer Society, Los Alamitos (2004)
12. Yao, Q., et al.: Finding and analyzing database user sessions. In: Zhou, L., Ooi, B.C., Meng, X. (eds.) DASFAA 2005. LNCS, vol. 3453, pp. 851–862. Springer, Heidelberg (2005)
13. Katz, S.: Estimation of probabilities from sparse data for the language model component of a speech recognizer. IEEE Transactions on Acoustics, Speech, and Signal Processing 35(3), 400–401 (1987)
14. Jelinek, F., Mercer, R.L.: Interpolated estimation of markov source parameters from sparse data. In: Pattern Recognition in Practice, pp. 381–397. North-Holland, Amsterdam (1980)
15. Dell Laboratories: Dell DVD store database test suite (2008), `http://linux.dell.com/dvdstore/`
16. Schnaitter, K., et al.: On-line index selection for shifting workloads. In: Proceedings of the 23rd International Conference on Data Engineering Workshops. IEEE Computer Society Press, Los Alamitos (2007)
17. Bruno, N., Chaudhuri, S.: An online approach to physical design tuning. In: Proceedings of the 23nd International Conference on Data Engineering, pp. 826–835. IEEE Computer Society Press, Los Alamitos (2007)
18. Elnaffar, S., Martin, P., Horman, R.: Automatically classifying database workloads. In: Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management. ACM Press, New York (2002)
19. Martin, P., et al.: Workload models for autonomic database management systems. In: Proceedings of the International Conference on Autonomic and Autonomous Systems, p. 10. IEEE Computer Society, Los Alamitos (2006)