The Psychic-Skeptic Prediction Framework for Effective Monitoring of DBMS Workloads

Said Elnaffar
College of IT
UAE University
Al-Ain, P.O. Box 17555
UAE
elnaffar@uaeu.ac.ae

Patrick Martin
Queen’s University
School of Computing
Kingston, ON K7L 3N6,
Canada
martin@cs.queensu.ca

Abstract
Self-optimization is one of the defining characteristics of an autonomic computing system. For a complex system, such as the database management system (DBMS), to be self-optimizing it should recognize properties of its workload and be able to adapt to changes in these properties over time. The workload type, for example, is a key to tuning a DBMS and may vary over the system’s normal processing cycle. Continually monitoring a DBMS, using a special tool called Workload Classifier, in order to detect changes in the workload type can inevitably impose a significant overhead that may degrade the overall performance of the system. Instead, the DBMS should selectively monitor the workload during some specific periods recommended by the Psychic-Skeptic Prediction (PSP) framework that we introduce in this work. The PSP framework allows the DBMS to forecast major shifts in the workload by combining off-line and on-line prediction methods. We integrate the Workload Classifier with the PSP framework in order to come up with an architecture by which the autonomous DBMS can tune itself efficiently. Our experiments show that this approach is effective and resilient as the prediction framework adapts gracefully to changes in the workload patterns.

1 Introduction

Commercial relational database management systems (DBMSs) are used for a range of applications from data warehousing through online transaction processing. As a result of this demand, DBMSs have continued to grow in terms of their size and the functionality they provide. This growth reciprocally creates additional overhead in the associated system complexity and administration costs [18] and makes the task of manually tuning the performance of these systems virtually impossible.
Autonomic computing systems, which are systems that are self-configuring, self-optimizing, self-healing and self-protecting, are a promising approach to solving these problems of cost and complexity [1]. In order to effectively tune a DBMS, the database administrator (DBA) must understand the workload placed on that system, which makes automatic workload characterization a prerequisite for automatic tuning. Our work focuses on the specific workload characterization problems of 1) identifying the type of a workload and 2) predicting changes in that type over time. This is because the type of the workload, specifically whether it is Online Transactional Processing (OLTP) or Decision Support System (DSS), is an important criterion for tuning [4][6][10]. Memory resources, for example, are allocated very differently for OLTP and DSS workloads. OLTP workloads contain a large number of small transactions that involve retrievals of individual records based on key values and updates. There are relatively few sorts and joins. DBAs therefore typically allocate memory to areas such as the buffer pools and log buffers while minimizing the sort heap. DSS workloads, on the other hand, contain a small number of large transactions that involve scans, joins and sorts. There are very few, if any, updates. DBAs therefore typically allocate more memory to the sort heap and the buffer pools. Once the dominant workload type is recognized and detected, the DBMS can adopt well known rule-of-thumb tuning strategies that experts found effective in practice (for examples of tuning strategies see [4][5][7][10]).

To that end, in [17] we developed an automatic workload classifier, briefly described below, for identifying the type of a workload put on a DBMS. However, a new challenge has been observed as the type of a workload tends to vary over time. Many organizations experience daily, weekly or monthly processing cycles that mirror the cycle of tasks performed by the organization. A bank, for example, may experience an OLTP workload during normal business hours, a DSS-like workload overnight while the previous day’s business is analyzed, and a heavy DSS workload at the end of the month when financial reports and executive summaries are issued. DBAs must therefore recognize the significant shifts in the workload and reconfigure the system accordingly in order to maintain acceptable levels of performance.

If we keep the workload classifier activated all the time in order to capture significant shifts in the type of the workload then we will impose undesirable overhead and perturbation on the system due to incessantly probing the DBMS engine for real-time statistics. This on-line analysis of the workload classifier and the constant monitoring may even undermine the benefits the workload classifier is supposed to contribute to the performance of the DBMS. By comparing the throughput of our DBMS, DB2, with and without using the workload classifier tool, we empirically found that running the workload classifier reduced the throughput of by 10% on average [17]1. Note that the workload classifier is a novel autonomic feature, however, if each new autonomic feature does not take care of reducing its operational cost, then the incremental introduction of features and functions could lead to accumulative overhead that threatens to undermine the very benefits autonomic computing aims to provide.

1 The exact overhead depends on the implementation of the tracking and monitoring tools associated with the DBMS under study.
1.1 Solution and Contribution

Therefore, and in order to mitigate such undesirable overhead, we suggest to predict when a change in the workload type will occur and limit the on-line monitoring to these specific time intervals instead of tracking the DBMS all the time. This is the main focus of this work.

Our key contribution in this paper is defined by presenting the Psychic-Skeptic Prediction (PSP) framework, which allows an autonomic DBMS to efficiently learn about a workload's dynamic behavior over time and to forecast when a change in the workload might occur in order to proactively reset the DBMS parameters to suit the new situation. The PSP framework is equipped by two categories of predictions: offline (long-term) prediction, and online (short-term) prediction.

Based on a comprehensive analysis for the entire workload, the offline prediction learns about the long-term pattern of the workload and, consequently, it can produce a complete schedule of anticipated shifts in the workload type. However, it is risky for the DBMS to base its reaction on mere prediction. As a safety net, the PSP framework counts on the online prediction to validate the forecasted shift at run-time. In addition to its reliability, the PSP framework adopts smart mechanism of well planned workload sampling in order to be adaptive and keep its prediction models up to date in case the workload starts to exhibit different patterns. The PSP framework determines all of its internal settings and decisions in such a way that optimizes the overall performance of the DBMS.

While we specifically demonstrate the effectiveness of the PSP framework in combination with the workload type classifier, we claim that our framework is generic in the sense that it can be used in other instances where workloads exhibit some trend that makes them relatively predictable. PSP could be used to automate many other DBMS tasks such as determining when to make incremental backups, re-build indexes and refresh materialized views, update statistics, or reorganize data on disk.

This remainder of the paper is structured as follows. We first review related work in autonomic DBMSs in Section 2. We then outline our previous work on workload classification in Section 3. Section 4 introduces our Psychic-Skeptic Prediction framework. Section 5 evaluates the performance of our framework using a set of simulation experiments. We summarize our paper and outline possible directions of future research in Section 6.

2 Autonomic Database Management Systems

DBAs face increasingly more difficult challenges brought about by the growing complexity of DBMSs, which stems from several sources [16]:

- Increased emphasis on Quality of Services (QoS). A DBMS must provide service guarantees in order that the overall system can satisfy the end-to-end QoS requirements.
• **Advances in database functionality, connectivity, availability, and heterogeneity.** DBAs must grapple with complex decisions about hardware platforms, schema design, constraints and referential integrity, primary keys, indexes, materialized views, the allocation of tables to disks, and shared-nothing, shared-everything, or SMP-cluster topology.

• **Ongoing maintenance.** Once designed, databases require substantial human input to build, configure, test, tune, and operate.

• **Burgeoning database size.** Popular applications such as SAP typically create more than 20,000 tables and support thousands of users simultaneously.

• **E-Service era.** Web-based applications present the DBMSs with a broader diversity of workloads in terms of type and intensity.

Autonomic computing systems, which shift the responsibility of managing the system from the system administrator onto the system itself, have emerged as a promising approach to dealing with this complexity [1]. A system is considered to be autonomic if it possesses several capabilities, namely if it is self-configuring, self-optimizing, self-healing and self-protecting. There has been a significant amount of research related to providing these capabilities in DBMSs.

The performance of a DBMS depends on the configuration of the hardware and software components. An autonomic DBMS should provide users with reasonable “out of the box” performance and dynamically adapt its configuration to provide acceptable, if not optimal, performance in light of constantly changing conditions. It must understand the composition and intensity of its workload in order to make the appropriate adjustments. Current DBMSs typically provide static configuration wizards to assist with initial configuration but offer little support for dynamic adjustment of resource settings. All the DBMSs provide an advisor (DB2’s Design Advisor [3], SQL Server’s Index Wizard [7], and Oracle’s Index Tuning Wizard [9]) to recommend a suitable set of indexes and materialized views.

Self-optimization is one of the most challenging features to include in a DBMS. It allows a DBMS to perform any task and execute any service utility in the most efficient manner given the present workload parameters, available resources, and environment settings. The system must understand the composition, intensity and priorities of the workload and be able to recognize shifts in the workload in order to effectively allocate resources. Work on this aspect has focused on issues such as the automatic collection and generation of operation statistics [8][13], more sophisticated cost models [7][8][11] and dynamic adjustment of query plans [24]. Our work in this research is a prerequisite for any optimization models that deem the characteristics of the workload as input parameters to these models. Many database internal parameters are set differently depending whether the workload is OLTP or DSS. OLTP business applications (such as PeopleSoft, Siebel, and SAP) support multiple users who require very rapid response times. Frequently, the database serves thousands of concurrent users. Response time may include CPU, sort, locking, and I/O times. The majority of SQL statements in an OLTP workload are **INSERT, UPDATE,**
and DELETE that require contention management and locking strategies. Yet, some OLTP applications include batch-processing components and probably some concurrent decision-support queries.

In contrast, DSS users ask complex business questions relevant to the available data requiring complex SQL queries. Response times in a DSS environment are typically measured in minutes rather than seconds. However, response time requirements vary significantly based on business needs. DSS workloads are mostly read-only queries. Parallelism (both CPU and I/O) greatly affects response times for these complex queries. OLTP jobs, on the other hand, are very small and efficient so parallelism (by which we mean intrapartition parallelism on a single SMP server) is neither necessary nor desirable.

These differences inevitably entail different settings of the configuration parameters in order to ensure that the DBMS has sufficient resources to perform the processing required. Detailed discussion on how to tune each database parameter (e.g., size of sort heap, number of I/O cleaners, log size, min. number of commits, etc) upon distinguishing whether the workload is OLTP or DSS can be found in [5] and the technical manuals of DBMSs (e.g., [4][7][10]).

A self-healing DBMS automatically detects, analyzes and repairs problems. It must understand the normal patterns and levels in the workload in order to detect possible sources of faults. The system must reliably log all operations, periodically archive the database and be able to use the logs and backups to recover from failure. Ideally, it should recognize when a full or incremental backup is necessary and perform these operations with minimal system disruption. In the event of catastrophic failure, a self-healing DBMS should be able to retrieve the most recent backup, restore to the consistent point just before the failure and then resume its halted operations after handling the exceptions. Oracle currently provides the ability to resume operations (such as a batch load) following corrective action (such as the addition of more disk space) [9]. DB2’s Recovery Expert is a recovery tool that analyzes the recovery assets available and recommends a technique to be selected. DB2’s Automatic Incremental Restore mechanism uses the backup history for automatically searching for the correct backup images.

A self-protecting DBMS automatically defends itself against malicious attacks. It must therefore understand the normal patterns and levels in the workload in order to detect possible security threats from outside sources. Database protection implies support for database security, privacy, analytical auditing mechanisms, data encryption, and admission control strategies. Current DBMSs all provide authentication and access control mechanisms. The DB2 Query Patroller [4] and the Oracle Resource Manager [9] are examples of admission control tools used today.

3 Workload Classification

The above examination of autonomic DBMSs reveals that workload characterization is an important prerequisite for all autonomic capabilities. Self-optimization, in particular, requires that the system be
Figure 1. The methodology of constructing a workload classifier

aware of several aspects of its workload, including composition, frequency patterns and intensity. As explained earlier, the type of a workload is one of the key factors used to tune a system. We have decomposed the problem of identifying the type of a workload into the two sub problems of classifying the workload and predicting when the type changes. The prediction problem is the focus of this paper. Our background work on classifying workloads is described elsewhere [14][15][17] and is summarized below.

We view the problem of classifying DBMS workloads as a machine-learning problem in which the DBMS must learn how to recognize the type of the workload mix. We use data mining classification techniques, specifically Decision Tree Induction [19], to build a classification model. One of the advantages of using decision tree induction is its high interpretability, that is, the ease of extracting the classification rules and the ability to understand and justify the results, in comparison with other techniques such as neural networks.

We base our analysis on dynamic data collected by DBMS performance monitors because the dynamic data includes frequency information that is not available from static sources, it captures the variability of
Figure 2. Workload Classifier

the workload over time, and it is easier to analyze than SQL statements or access plans. We first analyze attributes available from performance snapshots with respect to their suitability for differentiating between DSS and OLTP and derive a set of attributes to be used in our classification process. We next build a model (or classifier) to describe a predetermined set of data classes.

As shown in Figure 1, the model is constructed by analyzing a training set of data objects that constitute two thirds of the total data volume we have. Each object is described by attributes, including a class label attribute that tags the class of the object as either OLTP or DSS. The data objects needed to build the classifier are performance snapshots taken whilst the database processes the workload. Each snapshot reflects the workload properties at some time during the execution. The learned model is represented in the form of a decision tree embodying the rules that can be used to categorize future data objects.

In our experiments, we constructed workload classifiers from two pairs of representative workloads. The first pair consisted of the TPC-C [21] and TPC-H [22] benchmarks as the OLTP and DSS workloads,
respectively. TPC-C simulates a complete order-entry environment where a population of terminal operators executes transactions that include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses. It represents any industry that must manage, sell, or distribute a product or service. On the other hand, TPC-H is a decision support benchmark whose workload examines large volumes of data, executes queries with a high degree of complexity, and gives answers to critical business questions.

The second pair consisted of the Browsing and Ordering profiles defined in the TPC-W benchmark [23] as the DSS and OLTP workloads, respectively. TPC-W comprises a set of basic operations designed to exercise transactional web system functionality in a manner representative of internet commerce application environments. These basic operations have been given a real-life context, portraying the activity of a web site (bookstore) that supports user browsing, searching and online ordering activity. Visitor activities are described by three profiles: Browsing, Shopping, and Ordering. The Browsing profile is characterized by extensive browsing and searching activities. The Shopping profile exhibits some product ordering activities but browsing is still dominant. The Ordering profile has a majority of ordering activities. Therefore, the ultimate difference between these profiles is the browse-to-order ratio.

We ran each pair of workloads on IBM’s DB2 Universal Database system (DB2) [4] and obtained performance data for a variety of attributes using the DB2 Snapshot Monitor. For each pair of workloads we analyzed the data to obtain a minimal set of attributes that differentiated OLTP from DSS requests. The attributes selected for the TPC-W profiles, for example, included attributes such as the proportion of queries (versus updates) in the workload, the average number of pages read per request, the average number of rows selected by a request, the number of locks held by a request and the average sort time of a request. We found that attributes that are more independent of the system configuration, such as the queries ratio, the pages scanned and the rows selected, are more influential in determining the type of a workload than attributes that are dependent on the configuration. We therefore further grouped the attributes according to their dependence on the system configuration and assigned higher weights to the more independent groups. The classification model (decision tree) created for the TPC-W Browsing and Ordering profiles is shown in Figure 2. The DSSness, which is the metric reported by this DSS/OLTP workload classifier, represents the percentage of the DSS type versus OLTP. Thus, if DSSness is closer to its maximum value (100%) it means that the workload is mostly DSS; if it is closer to its minimum value (0%) it means that the workload is mostly OLTP. A middle value of DSSness indicates a mixed (MIX) workload. The goal of our prediction framework discussed below is to efficiently predict how the DSSness fluctuates over time.
The type of the workload typically changes over time but keeping the workload classifier tool running all the time in order to detect significant shifts in the workload type is not a feasible option. Our goal, therefore, is to propose an alternative, more efficient solution by which the DBMS can learn about a workload's dynamic behavior over time and forecast when a change in the workload type might occur in order to proactively reset the DBMS parameters to suit the new workload.

Figure 3 illustrates how the Workload Classifier and Workload Predictor integrate in order to provide an efficient workload identification solution. The workload classifier assesses the DSSness of the workload at a given time. The Workload Predictor, after it analyzes a time series of DSSness, forecasts major shifts in the DSSness and alerts the DBMS of these shifts. Major shifts are formed when the DSSness reaches predefined thresholds that warrant reconfiguring the DBMS. For the purposes of this paper the thresholds divide the DSSness range into three zones that lead to the identification of three main workload types: OLTP, MIX, and DSS.

4.1 Overview

Prediction techniques are typically classified as either on-line or off-line [12]. On-line prediction techniques are more accurate than off-line approaches but require continuous monitoring of a
system. Off-line prediction techniques, on the other hand, involve little overhead but are less accurate since they do not account for unexpected variations in a workload. The Psychic-Skeptic framework combines features of both on-line and off-line approaches to provide effective prediction. An initial prediction model is developed off-line. The model is then verified and refined using selective on-line monitoring of the system.

As depicted in Figure 4, the Psychic-Skeptic Predictor (PSP) consists of three main components: the Psychic, the Skeptic, and the TrainingDataModel. The Psychic analyzes a daily time series of DSSness values stored in the TrainingDataModel and produces an off-line prediction model, polynomial \( f(x) \), that can estimate major shifts in the DSSness with respect to some specified thresholds. The estimates are passed to the Skeptic which validates the shifts by performing on-line, short-term prediction using linear regression. If the Psychic advises that a shift is due at time \( t \), the Skeptic monitors and analyzes the system for some time interval centered around \( t \). The Skeptic does not instruct the DBMS to reset its parameters unless it confirms the trend of the shift using the linear model. Note that the Skeptic uses the linear regression as a short-term prediction model because: it is computationally a light-weighted tool to use at run-time, easy to discern the
trend by simply examining the sign of the slope of the line, and it only needs at least two sampled points in order to construct the model.

The Skeptic also performs regular sampling for the DSSness throughout the day in order to keep all prediction models up to date. It sends the results of the monitoring, which we call DSSness patching samples, to the TrainingDataModel in order to update the stored training data. The Psychic then refreshes its off-line prediction model and its forecasted shift schedule accordingly. This regular update guarantees the adaptability of the PSP and makes it less vulnerable to changes in the workload pattern. Without lack of generality, and for the sake of simplicity, we assume a day to be the time window over which our prediction framework operates. Therefore, the time scale consists of 1440 minutes (24 hours). However, the same concepts are applicable to any other time window such as weeks or months. We also assume the existence of some seasonality in the DSSness values over time.

### 4.2 PSP Parameters

The PSP framework exhibits a good degree of flexibility by counting on a number of configuration parameters that can be adjusted to suit the IT environment. Table 1 summarizes these parameters and categorizes them as either *global* or *TrainingDataModel-specific* parameters. Most of these parameters are automatically estimated by the PSP or obtained from the DBMS monitoring tools.

<table>
<thead>
<tr>
<th>Module</th>
<th>Parameter</th>
<th>Description</th>
<th>Range</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>MUM</td>
<td>Boolean parameter to activate or deactivate the Model Update Mechanism.</td>
<td>ON or OFF</td>
<td>ON</td>
</tr>
<tr>
<td></td>
<td>monCost</td>
<td>Workload classifier overhead</td>
<td>(0-100)%</td>
<td>10%</td>
</tr>
<tr>
<td></td>
<td>performanceMatrix</td>
<td>DBMS’s performance</td>
<td>0.0-1.0</td>
<td>See Table 2</td>
</tr>
<tr>
<td></td>
<td>dss_threshold</td>
<td>DSSness value that lies between the DSS and MIX workload zones</td>
<td>(0-100)%</td>
<td>70%</td>
</tr>
<tr>
<td></td>
<td>oltp_threshold</td>
<td>DSSness value that lies between the MIX and OLTP workload zones</td>
<td>(0-100)%</td>
<td>30%</td>
</tr>
<tr>
<td></td>
<td>min_check_time</td>
<td># minutes that Skeptic needs to validate a shift</td>
<td>30-1440</td>
<td>30 min.</td>
</tr>
<tr>
<td>TrainingDataModel</td>
<td>numScenarios</td>
<td># of daily scenarios used for training</td>
<td>1-5</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>numDaysToCompleteUpdate</td>
<td># of days needed to update a whole scenario</td>
<td>fastUpdateDays – slowUpdateDays</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 1. Psychic-Skeptic Predictor Parameters

The global parameters are used across several modules in the PSP and include the following:

**Model Update Mechanism (MUM)** is a switch parameter that can be set to ON or OFF to respectively enable or disable the Model Update Mechanism feature of the framework. In general, if the daily pattern is trusted to remain stable, the MUM parameter can be OFF. Setting MUM to ON entails a small overhead but guarantees that the prediction models are kept up to date and provides immunity against changes in the workload pattern.
\textbf{monCost} is the percentage of performance (throughput) degradation caused by running the workload classifier on-line. This percentage needs to be empirically determined once and remains constant afterwards.

\textbf{oltp\_threshold} and \textbf{dss\_threshold} determine the DSSness ranges for the three types of workload (DSS, OLTP and MIX):

\begin{equation}
\text{workloadType(DSSness)} = \begin{cases} 
\text{DSS} & \text{if DSSness > dss\_threshold} \\
\text{MIX} & \text{if oltp\_threshold \leq DSSness \leq dss\_threshold} \\
\text{OLTP} & \text{if DSSness < oltp\_threshold}
\end{cases}
\end{equation}

We empirically found that, in our experiments, 30 and 70 are good estimates for the \text{oltp\_threshold} and \text{dss\_threshold}, respectively.

\textbf{performanceMatrix} is a matrix of performance factors, each has a value range of 0 to 1.0. If \(x\) and \(y\) are workload types (OLTP, MIX or DSS), then the matrix entry \((x, y)\) denotes the relative performance of processing \(x\), while the DBMS is tuned to process type \(y\), to the performance of the DBMS, while it is tuned to process type \(x\) (i.e., the ideal tuning). These performance factors are empirically determined by running different combinations of workloads and DBMS settings. Table 2 shows the empirically estimated performance factors that we use in our experiments.

\textbf{min\_check\_time} is the minimum number of minutes needed by the Skeptic to execute on-line in order to validate the Psychic’s forecasted shift. This value determines the number of the DSSness samples that will be used to build the linear model at run-time. Throughout our experiments, we found that 30 minutes, which constitutes 2\% of the time scale of the day (\(max\_time\_scale=1440\) minutes), is a reasonable start. The final duration is eventually determined by the Psychic upon analyzing the training data.

### 4.3 The Training Data Model

The TrainingDataModel stores historical samples, or \textit{Scenarios}, to train the Psychic and the Skeptic. It maintains a number (\(num\_Scenarios\)) of chronologically ordered scenarios. Each scenario consists of a time series, \((t_i, DSSness_i)\), where \(DSSness_i\) is the DSSness reading reported by the workload classifier at time \(t_i\). Each scenario \(k\) \((0 \leq k \leq num\_Scenarios-1)\) is assigned a weight \(2^k\) that is normalized to the range \([0, 1]\), which ensures that the more recent observations have a larger impact than the older ones.

In order to verify the existence of a predictable, cyclic DSSness pattern over the \(num\_Scenarios\) days stored in the training model, we view the DSSness samples of all days as a single time series and then use

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
 & OLTP Settings & MIX Settings & DSS Settings \\
\hline
OLTP workload & 1.0 & 0.5 & 0.3 \\
MIX workload & 0.5 & 1.0 & 0.5 \\
DSS workload & 0.3 & 0.5 & 1.0 \\
\hline
\end{tabular}
\caption{Performance Matrix}
\end{table}
the autocorrelation coefficient, $r_k$, to test the predictability of the DSSness using the following formula [20]:

$$r_k = \frac{\sum_{i=1}^{N-k} (D_i - \overline{D})(D_{i+k} - \overline{D})}{\sum_{i=1}^{N} (D_i - \overline{D})^2}$$

(2)

where $k$ denotes the lag ($k=1,2,3,...$), which is the prediction timeframe ($k = \text{max\_time\_scale}$); $N$ is the total number of samples collected over a number ($\text{numScenarios}$) of days, $N = \text{numScenarios} \times \text{max\_time\_scale}$; $D_i$ is the DSSness sample number $i$, where $i = 1..N$; $\overline{D}$ is the mean of the DSSness samples, that is,

$$\overline{D} = \frac{\sum_{i=1}^{N} D_i}{N}$$

The range of the autocorrelation coefficient $r_k$ is [-1, 1]. A near-zero value indicates a lack of correlation between the DSSness values occurring at the same time within each day. A positive value of $r_k$ indicates a conformance of the DSSness trend while a negative value indicates an inverse trend. Therefore, a larger positive $r_k$ means that our workload prediction methodology is more applicable due to the existence of a predictable pattern in the given workload. In our experiments, we found that $r_k$ is 0.65 on average.

For the computation purpose, the rest of the PSP components work deal with a consolidated scenario that summarizes all the information from the training model as a single average day. The consolidated scenario is constructed by calculating the weighted average of all samples collected at time $t$ across all scenarios:

$$\text{avgDSSness}_t = \frac{\sum_{d=0}^{\text{numScenarios}-1} DSSness(d,t) \times W_d}{\sum_{d=0}^{\text{numScenarios}-1} W_d}$$

(3)

where $W_d = \frac{2^d}{\sum_{i=0}^{\text{numScenarios}-1} 2^i} = \frac{2^d}{2^{\text{numScenarios}} - 1}$, and denotes the weight of scenario number $d$.

The PSP adapts to pattern changes using a Model Update Mechanism (MUM) that patches the training data. Patching is a function by which the Skeptic can gradually update the historical data in order to keep the underlying prediction models up to date. It occurs by propagating the DSSness samples of a particular scenario to the next older scenario. A DSSness$_{(d, t)}$ sample of day $d$ at time $t$ replaces DSSness$_{(d-1, t)}$ for $d=1..\text{numScenarios}-2$. DSSness$_{(\text{numScenarios}-1, t)}$, which is the most recent scenario in the TrainingDataModel, is patched by the new DSSness samples collected by the Skeptic.
The goal of this algorithm is to identify all possible shifts that can occur during the day and store them in the shiftSchedule data structure.

```
ALGORITHM findShifts(output: shiftSchedule)
BEGIN
Create a new blank shiftSchedule;

For each threshold, ∀threshold ∈ {dss_threshold,oltp_threshold}
   {  
      candidateShifts = the points where the DSSness curve intersects with threshold
   }
   For each candidateShift, s ∀s ∈ candidateShift
   {
      slope = first derivative at s;

      // Filter out minima and maxima shifts because they barely touch the threshold
      If(slope is not near zero) //i.e., not minima or maxima
      {
         Create a new shift shift;
         time of shift = s;
         If(threshold = dss_threshold)
         {
            If (slope >0)
               Type of shift = MIX_UP_TO_DSS;
            Else
               Type of shift = DSS_DOWN_TO_MIX;
         }  
         If(threshold = oltp_threshold)
         {
            If (slope >0)
               Type of shift = OLTP_UP_TO_MIX;
            Else
               Type of shift = MIX_DOWN_TO_OLTP;
         }
         Add shift to shiftSchedule;
      } // if not minima or maxima
   } // for each shift
} // for each threshold

END
```

Figure 5. Finding Shifts

Selecting the value of numScenarios is a tradeoff between the quality of the off-line model and the pace by which the entire stack of scenarios in the data set can be fully refreshed. A large number of days in the training set makes the PSP framework less vulnerable to isolated changes or spikes in the workload behavior (e.g., generating an unscheduled annual report). On the other hand, with a large value of
numScenarios, the PSP becomes less adaptive to changes in the workload pattern (i.e., PSP will require a longer time to discern the observed changes as a new pattern).

Experimentally, we found that numScenarios=3 is a reasonable size that produces good prediction quality and high adaptability.

4.4 The Psychic

The Psychic is primarily responsible for producing an off-line prediction model by tapping the cyclic pattern that occurs during the day. More specifically, it carries out five main tasks in the following sequence:

1. **Off-line Model Construction.** The Psychic analyzes historical data stored in the TrainingDataModel and produces a polynomial model to fit the data.
2. **Finding Shifts.** The polynomial model is used to find the potential workload shifts by finding intersection points of this polynomial with the dss_threshold and oltp_threshold.
3. **Estimating Shift Check Time.** In order for the Skeptic to validate a particular shift that has been forecasted by the Psychic, the Skeptic needs a timeframe during which it monitors the workload and eventually decides whether to endorse this shift or to disregard it. The Psychic estimates the start and end times of the timeframe for each shift.
4. **Filtering Shifts.** The Psychic performs a cost-benefit analysis to determine if the shift is worth the consideration or if ignoring it is more beneficial to overall system performance.
5. **Configuring the MUM Parameters.** The MUM assures the validity of prediction models used in this framework by making them less vulnerable to possible changes in the workload pattern over time.

4.4.1 Off-line Model Construction

There are many tools that can be used for time series prediction such as neural networks, ARMA/ARIMA (Autoregressive Moving Average/Autoregressive Integrated Moving Average) models, DPLL (Digital Phase Locked Loop), digital filters, or Fourier series [20]. These models can be used to predict the DSSness (dependent variable) at a given time (independent variable). However, the Psychic needs to predict when (i.e., time) the DSSness reaches a specific threshold. This requires dealing with the inverse of the prediction function, which is not always easy to derive using the above prediction tools. The Psychic therefore uses polynomial regression, which allows for straightforward geometric manipulation (i.e., it is easy to find where the polynomial intersects with certain thresholds) and is an intuitive, compact representation of the workload trend. In our experiments, the off-line models are mostly 3rd and 4th degree polynomials.

4.4.2 Finding Shifts

The Psychic uses the generated polynomial model to find the times at which the DSSness index intersects with the dss_threshold or the oltp_threshold by calculating the roots of the polynomial f(t) when f(t) –
Figure 6. Shifts form when the DSSness index intersects with the thresholds.

\[ dss\_threshold = 0 \text{ and } f(t) - oltp\_threshold = 0 \]. We must be careful to exclude the points that are minima and maxima as they almost touch the threshold levels and do not actually embody real shifts. These false shifts are identified by checking the slope of the curve using the first derivative, \( f'(x) \). If \( f'(t) \approx 0 \), then shift \( t \) must be discarded.

A shift can be one of four types depending on its trend: OLTP_UP_TO_MIX, MIX_UP_TO_DSS, DSS_DOWN_TO_MIX, or MIX_DOWN_TO_Oltp. The slope of the shift can determine the direction of the shift by identifying its inclination (slope \( > 0 \)) or declination (slope \( < 0 \)). Figure 5 sketches the algorithm for detecting shifts.

### 4.4.3 Estimating Shift Check Time

The Psychic must also determine the shortest period of time during which the Skeptic must run to validate a forecasted shift. This goal is achieved in two steps:

1. **Determining Shift Bounds.** The extreme bounds that delimit a shift are determined by the nearest local maximum and local minimum surrounding the shift time as illustrated in Figure 6. They represent the search space, \( [a..b] \), for estimating the earliest\(CheckTime\) and latest\(CheckTime\) period (denoted as \( e \) and \( l \) respectively in the figure), as explained in step 2 below. The first and last shifts may become special cases. If the first shift is not preceded by a minimum or maximum, the lower\(BoundCheckTime\) is set to zero, which is the beginning of the day. If the last shift is not followed by any minimum or
/* The goal of this algorithm is to determine the maximum checking limits for each shift from the left (lowerBoundCheckTime) and the right (upperBoundCheckTime).
*/

ALGORITHM DeterminingShiftBounds(input: shiftSchedule)
BEGIN
Store all local minima and maxima in list localMAndM[]
numLocalMAndM = size of localMAndM[]

For each shift[i], 0 <= i <= numShifts
{
    found = false;

    For each localMAndM[i], 0 <= i <= numLocalMAndM
    {
        If (localMAndM[i] > shift[i].time)
        {
            // Handling the FIRST shift’s special case
            If (i=0)
                shift[i].lowerBoundCheckTime(0); // the start of the day
            Else
                shift[i].lowerBoundCheckTime = localMAndM[i-1];

                shift[i].upperBoundCheckTime = localMAndM[i];
                found = true;

            ExitLoop; // terminate the search for localMAndM
        } // if
    } // for each minima and maxima

    // If no minima or maxima follows a shift
    If (NOT found)
    {
        shift[i].upperBoundCheckTime = MAX_TIME_SCALE;

        // Setting the lower bound
        If (numLocalMAndM > 0)
            shift[i].lowerBoundCheckTime = localMAndM[numLocalMAndM - 1];
        Else
            shift[i].lowerBoundCheckTime = 0;
    } // if
} // for

END

Figure 7. Determining Shift Bounds

2. maximum then the upperBoundCheckTime is set to the last minute of the day (max_time_scale). Figure 7 describes how to find the [lowerBoundCheckTime, upperBoundCheckTime] period.

3. Estimating Earliest and Latest Check Times. In this step the Psychic tries to find a subset period, [earliestCheckTime, latestCheckTime], within the [lowerBoundCheckTime, upperBoundCheckTime] of a shift. This is imperative as it reduces the time needed by the Skeptic to validate a shift at run time.
/*
The goal of this algorithm is to find the smallest period of time around each shift that is adequate for the on-line Skeptic to confirm the trend (predetermined by the Psychic) of that shift
*/

ALGORITHM EstimatingEarliestAndLatestCheckTimes
BEGIN
For each shift[i], 0<=i<=numShifts {
shift[i].latestCheckTime = shift[i].time+(min_check_time/2);
shift[i].earliestCheckTime = shift[i].time-(min_check_time/2);
// All experimentation is done on the consolidated scenario
While [earliestCheckTime, latestCheckTime] ⊆ [lowerBoundCheckTime, upperBoundCheckTime] for each shift[i]
// The skeptic is used to compute to obtain the linear model
slope = linear slope at shift[i] within time period [earliestCheckTime, latestCheckTime]
// trendMatches is a Boolean expression that determines whether the pre-known shift trend, as was seen by the psychic, conforms with the linear model as seen by the skeptic

  trendMatches = (shift[i].isTrendingUp AND slope >0) OR (shift[i].isTrendingDown AND slope <0)
If (trendMatches)
  exitLoop;
Else {
  // Enlarge the checking period from left and right as long they do not hit bounds
  If (shift[i].earliestCheckTime > shift[i].lowerBoundCheckTime)
    shift[i].earliestCheckTime = shift[i].earliestCheckTime -1
  If (shift[i].latestCheckTime < shift[i].upperBoundCheckTime )
    shift[i].latestCheckTime = shift[i].latestCheckTime +1
  // else

  } // Loop
  // for each shift
END

Figure 8. Estimating earliest and latest check times of a shift

Figure 8 describes how to estimate [earliestCheckTime, latestCheckTime] by analyzing the training scenarios stored in the TrainingDataModel. Initially, the Psychic starts with earliestCheckTime=t-(min_check_time/2), and latestCheckTime=t+(min_check_time/2), where t denotes the expected shift time. This interval is incrementally expanded until the Skeptic’s linear model applied to the consolidated scenario agrees with the trend of the shift. Expansion is performed by decrementing earliestCheckTime and incrementing latestCheckTime such that the conditions earliestCheckTime ≥ lowerBoundCheckTime and latestCheckTime ≤ upperBoundCheckTime are not violated.
The goal of this algorithm is to estimate the values of the MUM parameters: slowUpdateDays and fastUpdateDays, along with their associated expected performance slowUpdatePerf and fastUpdatePerf, respectively.

```cpp
ALGORITHM SetMUMParameters
BEGIN
    // all tests are applied to the consolidated scenario
    s = consolidatedScenario

    p1 = performance using out-of-the-box settings that suit the dominant workload type.
    p2 = performance using Continuous Monitoring

    // Maximum # of days to update the model implies checking only once a day
    maxUpdateDays = ceil(max_time_scale / min_check_time);

    For each i, 1<=i<= maxUpdateDays
    
        daysToCompleteModelUpdate = i
        p3 = performance using the PSP framework

        If (p3 > (p1+PERF_GAIN) AND p3 > (p2+PERF_GAIN))
        
            fastUpdateDays = i
            fastUpdatePerf = p3
            exitLoop;
        
    // Now, we set the MUM to work conservatively (slowly). The user can opt to higher speeds later.
    slowUpdateDays = maxUpdateDays;
    daysToCompleteModelUpdate = slowUpdateDays
    slowUpdatePerf = performance using the PSP framework

END
```

**Figure 9. Estimating MUM parameters**

### 4.4.4 Filtering Shifts

Reacting to some of the detected shifts might not be beneficial to system performance. For example, a shift might be too short and it is not cost-effective to reset the DBMS’s configuration parameters. The Psychic performs a cost-benefit analysis for each shift in order to decide whether to accept or reject a shift. This decision is made by estimating the performance difference between the two cases where the shift is accepted and where it is discarded. Note that filtering is not needed if we have fewer than two shifts. The cost-benefit computation analysis is ultimately based on the global parameters performanceMatrix and monCost.
4.4.5 Estimating Model Update Mechanism (MUM) Parameters

The MUM aims to ensure that all prediction models are up to date, which makes the prediction accuracy less vulnerable to changes in the daily pattern. It achieves this goal by performing regular DSSness sampling for short, carefully selected intervals throughout the day.

The MUM views a day as a number of equal, ordered time zones, where each zone is further divided into a number of equal, ordered time slots. Given \( n \) slots in a zone, the Skeptic ensures uniform coverage of the daily workload by monitoring over a cycle of \( n \) days where slot \( i \) (\( 0 \leq i \leq n-1 \)) of each zone is monitored on day \( i \) of the cycle.

At run-time, the Actuator, a subcomponent in the Skeptic, monitors the system clock for the start time \( S \) and end time \( E \) of the upcoming slot in order to ask the Skeptic to start and stop sampling at these times, respectively. \( S \) and \( E \) are calculated as follows:

\[
S = Day \times SlotSize + Zone \times ZoneSize
\]

\[
E = S + SlotSize
\]

(4)

\( Day \) denotes the day number throughout the model-update process. Initially, \( Day \) is set to 0 in the first day and it is incremented at the end of each day. \( Zone \) denotes the current zone number. It is reset to 0 at the beginning of each day and is incremented at the end of each slot. \( SlotSize \) and \( ZoneSize \) are constants denoting the size of the slot and the zone respectively, where:

\[
SlotSize = \text{min\_check\_time}
\]

(5)

\[
ZoneSize = SlotSize \times \text{numDaysToCompleteUpdate}
\]

(6)

\[
\text{numZones} = \left\lceil \frac{\text{max\_time\_scale}}{ZoneSize} \right\rceil
\]

(7)

We set \( SlotSize \) to \( \text{min\_check\_time} \), which is the shortest check period that the Skeptic uses to validate a shift. \( \text{numDaysToCompleteUpdate} \) is a parameter that specifies the number of days by which the first scenario in the training set is fully refreshed. Its value ranges from \( \text{fastUpdateDays} \) to \( \text{slowUpdateDays} \), where \( \text{fastUpdateDays} \) denotes the minimum number of days that the MUM needs in order to complete the update while the performance of the framework remains superior, and \( \text{slowUpdateDays} \) is the maximum number of days needed to update the model such that one slot is sampled every day. It is easy to realize that
slowUpdateDays = \left[ \frac{\text{max time scale}}{\text{SlotSize}} \right]. \text{fastUpdateDays} \text{ is optimized by starting by one day and incrementing it until the prediction framework outperforms the other operation modes (Section 5) by an arbitrary percentage \text{PERF_GAIN}, that is, } \text{fastUpdatePerf} >= (\text{anyOtherPerf} + \text{PERF_GAIN}). \text{ We use } \text{PERF_GAIN}=1\% \text{ in our experiments. fastUpdatePerf and slowUpdatePerf are the performance measures associated with setting } \text{numDaysToCompleteUpdate} \text{ to fastUpdateDays and slowUpdateDays respectively. Therefore, } \text{numDaysToCompleteUpdate} \text{ creates a tradeoff between the pace at which the framework can fully update a training scenario and the performance level. Setting } \text{numDaysToCompleteUpdate} \text{ to fastUpdateDays leads to a faster update, but with a relatively lower performance due to the incurred run-time monitoring. Setting } \text{numDaysToCompleteUpdate} \text{ to slowUpdateDays leads to the maximum DBMS performance but a longer time is required to update the model. In general, the framework has the ability to estimate the DBMS’s performance for any value assigned to } \text{numDaysToCompleteUpdate} \text{ and vice versa. Figure 9 gives the algorithm to estimate the MUM parameters and Table 3 summarizes the variables used in the algorithm.}

### 4.5 The Skeptic

Fortunately, we almost covered all technical aspects that explain how the Skeptic works through the discussion of the details of the PSP architecture in the sections above. However, it is worth reminding the reader in this final subsection of main functions of the Skeptic.

As we have seen, the Skeptic is responsible for validating the Psychic’s forecasted shifts. For each upcoming shift, the Skeptic samples the workload from \text{earliestCheckTime} to \text{latestCheckTime}. The workload samples are analyzed to confirm whether the trend of a shift conforms to the Psychic’s prediction. To that end, the Skeptic builds an on-line prediction model using linear regression that fits the collected samples. The slope of the line is used to determine the trend of the workload at run time. If the on-line
prediction model confirms the shift, the DBMS’s settings are reset to suit the upcoming workload type. Otherwise, the DBMS resets its settings to the default, which is the safest resort and can sub-optimally handle MIX workloads of OLTP and DSS.

The Skeptic has also another vital functionality that warrants that the prediction models of the PSP architecture are not stale. Whether the MUM is enabled or disabled, the Skeptic samples collected during the validation process, which are deemed the most recent observations of the day during that interval, are passed on to the MUM component in order to patch the historical data stored in the TrainingDataModel.

5 Evaluation

Our evaluation has two main goals. First, we validate the Psychic-Skeptic prediction framework\(^2\) by comparing the performance of a DBMS running under the framework with the performance of that system running in alternative operation modes. Second, we show that our approach is robust and able to adapt to changes that may occur in the workload pattern.

In our evaluation, we assume that a DBMS can run in one of the following operation modes:

1. **Out-of-the-box (Default) Mode.** This is a trivial operation mode in which the DBA chooses to run the DBMS with out-of-the-box default settings that suit a mixed workload. These settings remain static and do not respond to any changes in the workload type nor adapt to the dominancy of a particular workload type.

2. **Dominant Workload Mode.** The consolidated scenario obtained by the TrainingDataModel is analyzed in order to determine the dominant workload type. This is done by measuring the total time (in minutes) that each workload type lasts throughout the day. The workload type that runs for the longest accumulated time is deemed dominant, and the DBMS is configured to suit this dominant workload. This configuration is static and does not change. The performance obtained from this mode is always expected to outperform the Default Mode described above. However, it is not expected to provide the best performance as it is not adaptable.

3. **Continuous Monitoring Mode (MA).** This mode takes full advantage of the Workload Classifier by performing on-line, short-term prediction using the moving average (MA) \(^2\)

\[
d_{t+1} = \frac{y_t + y_{t-1} + \ldots + y_{t-n+1}}{n}
\]  

where \(d_{t+1}\) is the DSSness forecast value for the period \(t+1\), \(y_t\) is the actual value (observation) at time \(t\), and \(n\) the number of DSSness samples used to calculate \(d_{t+1}\). One problem with this technique

\(^2\) The Psychic-Skeptic Framework is implemented in Java.
is the determination of \( n \), the number of periods included in the averaging. \( n \) should be selected such that it minimizes the forecasting error, which is defined by the square of the difference between the forecast and actual values. The mean squared error (MSE) is given by

\[
MSE = \frac{\sum_{t=1}^{n}(y_t - d_t)^2}{n}
\]  

Different values of \( n \), such that \( MAX >= n >=1 \), may be tested using the historical data to find the one that gives the smallest MSE. We arbitrarily use \( MAX=10 \) as a maximum value for \( n \). Detecting a shift using a single MA value leads to instability as the value may oscillate around the threshold lines \( oltp\_threshold \) and \( dss\_threshold \). To avoid this pitfall, we use a timer to ensure that all forecasts satisfy the threshold for the last \( min\_check\_time/2 \) minutes. \( min\_check\_time \) is the initial check time needed by the Skeptic to validate shifts by sampling before and after the expected shift time. In this MA technique we need to check before the expected start time of the shift, and therefore, we use half of the interval.

Continuous mode is advantageous as it is responsive to changes in the workload. However, it involves the undesirable, on-line overhead of running the Workload Classifier. In general, this mode is very adaptive but costly.

4. **Psychic-Skeptic Mode.** The use of the classifier is governed by the PSP framework.

5.1 **Experimental Setup**

We test our framework using artificially generated data, which give us the flexibility of examining any hypothetical workload pattern we want to investigate even if it is unlikely to happen in practice. DSSness data are generated using the notion of Scenarios and ScenarioDescriptors. A ScenarioDescriptor can be perceived as a template for generating many scenarios that exhibit a particular daily pattern. A ScenarioDescriptor consists of a set of pairs \((time, DSSness)\) that play the role of anchors of DSSness values on the final DSSness curve. \( time \) is a minute during the day so its domain is \([0, 1440]\) (24 hours a day), and \( DSSness \) ranges from 0 to 100. These anchors enable us to direct and shape the trend of the DSSness in any way we desire. In order to generate a scenario out of this descriptor, a series of DSSness values are automatically generated between every two consecutive anchors. In order to make our scenarios more realistic, we inject a \( \pm (0\text{-}5)\% \) of random noise in the \( DSSness \), and \( \pm (0\text{-}2)\% \) of random noise in the \( time \), which is equivalent to \( \pm (0\text{-}30) \) minutes. Noise injected in the \( time \) dimension affects when a shift may start or end. Noise injected in the \( DSSness \) dimension affects whether a shift is likely to occur or not, based on its intersection with the threshold lines.

In each experiment, we simulate the execution of a DBMS run under each of the four operation modes for 30 days. The performance of the Psychic-Skeptic framework is evaluated when the MUM is enabled and when it is disabled. We use the default parameter settings described in Table 1, unless otherwise
indicated. We report both absolute performance and relative performance. By absolute performance, we mean the performance achieved as a percentage of the theoretical maximum performance that can be achieved by matching the DBMS settings with the workload type on a minute-by-minute basis. The maximum theoretical performance is obtained by extrapolating upon the results of empirical observations of the response time performance of a real DBMS under real workloads and configurations. By the relative performance we refer to the percentage of performance improvement (or degradation) of any operation mode with respect to the Default Mode.

5.2 Experiment 1: Pattern A

Figure 10 shows an instance scenario of the daily pattern (pattern A) used in the first experiment. The workload is mostly OLTP in the first two hours of the day; it changes to a mixed workload over the next 12 hours; it returns to a dominant OLTP workload for the next 9 hours and then shifts back to a mixed workload. The autocorrelation coefficient, $r_k$, of this workload = 0.6562, which indicates a predictable cycle of DSSness across multiple days.
The Psychic’s off-line prediction model for this daily pattern, which is the solid line shown in Figure 12, is

\[ f(x) = -6.61 + 0.32X -4.07E-4X^2 + 2.64E-8X^3 + 8.73E-11X^4 \]

The initial shift schedule for this daily pattern is shown in Table 4. We note that the mixed workload is dominant for pattern A so the performance obtained under the Dominant Mode is equivalent to the Default Mode. Table 5 summarizes the performance statistics observed with the Pattern A workload using the four operation modes, while MUM is ON and OFF.

Figure 11 shows the DBMS’s absolute performance under different operation modes when the MUM is off. The best performance is achieved under the Psychic-Skeptic framework (avg. 97.08%), followed by the MA Mode (avg. 87.39%), then the Dominant Mode (avg. 76.23%), which is equivalent to Default Mode. Figure 12 shows DBMS’s relative performance when the MUM is off. The Psychic-Skeptic framework achieved an average of 27.37% performance improvement over the Default Mode compared to 14.65% performance improvement achieved by the MA Mode. All performance estimates are based on workload classifier overhead of 10% (\(\text{monCost} = 10\%\)).

<table>
<thead>
<tr>
<th>Shift Type</th>
<th>EarliestCheckTime</th>
<th>Time</th>
<th>LatestCheckTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift 1: OLTP_UP_TO_MIX</td>
<td>152</td>
<td>167</td>
<td>182</td>
</tr>
<tr>
<td>Shift 2: MIX_DOWN_TO_OLTP</td>
<td>804</td>
<td>819</td>
<td>834</td>
</tr>
<tr>
<td>Shift 3: OLTP_UP_TO_MIX</td>
<td>1278</td>
<td>1293</td>
<td>1308</td>
</tr>
</tbody>
</table>

Table 4: Shifts of pattern A
Figures 13 and 14 show the performance the DBMS achieved in PSP mode with the MUM turned on to
\textit{numDaysToCompleteUpdate} = 7, that is, a complete daily scenario will be updated after one week. Figure 13 shows that the DBMS in PSP mode achieves the best performance (avg. 94.54%), followed by the MA Mode (avg. 87.37%), then the Dominant Mode (76.23%), which is equivalent to Default Mode. The absolute performance achieved in PSP mode with the MUM on is only slightly less than the performance achieved with the MUM off. This indicates that the additional overhead required by the MUM, which ensures that the system can adapt to changes in the workload pattern, is not excessive. Figure 14 shows the relative performance of the PSP mode with MUM enabled. We see that the PSP mode achieves an average

<table>
<thead>
<tr>
<th>MUM</th>
<th>Perf.</th>
<th>Statistic</th>
<th>Default</th>
<th>Dominant</th>
<th>MA</th>
<th>PSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>Absolute</td>
<td>Mean</td>
<td>76.23</td>
<td>76.23</td>
<td>87.39</td>
<td>97.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Std. Dev.</td>
<td>0.82</td>
<td>0.82</td>
<td>0.26</td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conf. Int.</td>
<td>[75.95, 76.51]</td>
<td>[75.95, 76.51]</td>
<td>[87.30, 87.48]</td>
<td>[96.90, 97.26]</td>
</tr>
<tr>
<td></td>
<td>Relative</td>
<td>Mean</td>
<td>n/a</td>
<td>0</td>
<td>14.65</td>
<td>27.37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Std. Dev.</td>
<td>n/a</td>
<td>0</td>
<td>1.24</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conf. Int.</td>
<td>n/a</td>
<td>n/a</td>
<td>[14.22, 15.08]</td>
<td>[26.99, 27.75]</td>
</tr>
<tr>
<td>ON</td>
<td>Absolute</td>
<td>Mean</td>
<td>76.23</td>
<td>76.23</td>
<td>87.37</td>
<td>94.54</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Std. Dev.</td>
<td>0.82</td>
<td>0.82</td>
<td>0.29</td>
<td>1.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conf. Int.</td>
<td>[75.94, 76.51]</td>
<td>[75.94, 76.51]</td>
<td>[87.27, 87.47]</td>
<td>[94.08, 95.01]</td>
</tr>
<tr>
<td></td>
<td>Relative</td>
<td>Mean</td>
<td>n/a</td>
<td>0</td>
<td>14.63</td>
<td>24.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Std. Dev.</td>
<td>n/a</td>
<td>0</td>
<td>1.25</td>
<td>2.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Conf. Int.</td>
<td>n/a</td>
<td>n/a</td>
<td>[14.19, 15.06]</td>
<td>[23.30, 24.75]</td>
</tr>
</tbody>
</table>

Table 5: The DBMS’s performance under pattern A
Figure 11. Absolute performance for pattern A (MUM is off).³

Figure 12. Relative performance for pattern A (MUM is off).

performance improvement of 24.03% over the Default Mode compared to 14.63% achieved by the MA Mode.

³ The series of Default are concealed behind the series of the Dominant because the Default and Dominant workloads coincide in these experiments.
Figure 13. Absolute performance for pattern A (MUM is ON)\(^4\).

Figure 14. Relative performance for pattern A (MUM is ON).

\(^4\) The series of Default are concealed behind the series of the Dominant because the Default and Dominant workloads coincide in these experiments.
5.3 Experiment 2: Pattern B

Figure 15 shows an instance scenario of another workload pattern (Pattern B). The DBMS experiences a workload that is mostly mixed in the first 8 hours of the day. The workload becomes more DSS over the next 10 hours. In the next 6 hours, it seems to be more of a MIX, and then it shifts to an OLTP for the rest of the day. The autocorrelation coefficient \( r_k \) is 0.6628, which indicates a predictable cycle of DSSness across multiple days.

The Psychic’s off-line prediction model for this daily pattern is

\[
f(x) = 52.53 - 0.12X + 7.71\times 10^{-4}X^2 - 9.98\times 10^{-7}X^3 + 3.55\times 10^{-10}X^4
\]

The initial shift schedule for this daily pattern is shown in Table 7. The dominant workload for pattern B is MIX, therefore, the performance under the Dominant and the Default modes is equivalent. Table 6 summarizes the performance statistics observed with Pattern B while MUM is ON and OFF, using the four operation modes.
<table>
<thead>
<tr>
<th>Shift Type</th>
<th>EarliestCheckTime</th>
<th>Time</th>
<th>LatestCheckTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift 1</td>
<td>MIX_UP_TO_DSS</td>
<td>358</td>
<td>373</td>
</tr>
<tr>
<td>Shift 2</td>
<td>DSS_DOWN_TO_MIX</td>
<td>914</td>
<td>929</td>
</tr>
<tr>
<td>Shift 3</td>
<td>MIX_DOWN_TO_OLTP</td>
<td>1178</td>
<td>1193</td>
</tr>
</tbody>
</table>

Table 7. Shifts for pattern B
Figure 16. Absolute performance under pattern B (MUM is off). The series of Default are concealed behind the series of the Dominant because the Default and Dominant workloads coincide in these experiments.
Figures 18 and 19 compare the performance of the DBMS under the four operation modes when the PSP mode has the MUM turned on with $numDaysToCompleteUpdate = 7$, that is, a complete daily scenario is updated after one week. PSP mode, as shown in Figure 18, still outperforms the others (avg. 94.14%), followed by the MA Mode (avg. 87.74%), then the Dominant Mode (74.04%), which is equivalent to the Default Mode. The additional overhead caused by the MUM is again relatively small for Pattern B. Figure 19 shows that the PSP mode achieves an average performance improvement of 27.24% over the Default Mode compared to 18.50% achieved by the MA Mode. The Skeptic invalidates the shift that is supposed to occur at minute 933 in the sixth and 20th days. These incorrectly predicted shifts cause no severe impact on the performance. However, the Skeptic invalidates the shift that is supposed to occur at minute 1201 in the 9th day. As seen in Figure 18, this prediction error has a significant impact on the overall performance in that day.

Figure 17. Relative performance for pattern B (MUM is off).
5.4 Adaptability: Pattern A changes to Pattern B

The goal of this experiment is to demonstrate one of the vital features of the PSP framework, namely its adaptability to changes in the daily pattern. We run the DBMS under pattern A, described above, for 30 days. We then swiftly switch to pattern B, under which the behavior of the DBMS is examined for another 6 days.

The series of Default are concealed behind the series of the Dominant because the Default and Dominant workloads coincide in these experiments.
30 days. This sudden shift in the daily pattern is unrealistic as changes usually happen gradually over several days. We advertently made such a swift change in order to push the framework to its limits and observe the worst case scenario. The MUM is active all the time and numDaysToCompleteUpdate is set to 7.

The performance of the DBMS during the first month is assessed with respect to the expected performance under pattern A. Similarly, the DBMS performance during the second month is assessed with respect to the expected performance under pattern B. The results indicate how much the observed DBMS performance deviates from the mean of the expected performance. Since the means of the two performance indices can be different, we normalize the differences in the performance by expressing them as percentages, as follows:

\[ \text{perfDiff} = \left( \frac{\text{Observed} - \text{Expected}}{\text{Expected}} \right) \times 100 \] (10)

Figure 20 shows the percentage of the performance deviation from the expected one. During the first 30 days as the DBMS handles workload of pattern A, the mean observed performance is 94.50%, as compared to the expected performance of 94.54%. At the end of the first 30 days processing pattern A, we expose the DBMS to a new workload matching pattern B. On day 31, the performance dramatically drops to 56.3% of the expected performance (94.19%), which is a 40% decrease in performance. With the MUM enabled, however, the performance gradually improves over the 7 day updating period. By the end of this period, the performance reaches 93.90%. The mean performance over the period ending the update to the
end of the second month (i.e., 21 days) is 93.57%, which is close to what we expect, 94.19%. This shows that the system is able to return to its stable state.

6 Summary

Systems monitoring and constant on-line analysis incur performance penalties that may hinder the adaptation of tools such as the workload classifier described in this paper. Luckily, the overhead of such tools can be mitigated by exploiting characteristics in the workload. In this paper, we introduce the Psychic-Skeptic prediction (PSP) framework. The Psychic analyses historical data and produces a shift schedule. Each shift indicates whether the workload is heading to the DSS, OLTP, or MIX region. These regions are delimited by two DSSness thresholds. The DBMS does not place its full trust in the off-line predicted shifts and asks the Skeptic to validate each shift at run-time by sampling the workload for a small interval around the expected shift time in order to confirm its direction. If the shift is approved, the DBMS resets its configuration to suit the new workload type.

The framework is self-optimizing as the majority of its parameters are automatically estimated and are transparent to the database administrator. The framework adapts to changes in the workload pattern using the Model-Update-Mechanism (MUM), which samples the workload at regular times in order to patch historical data and keep prediction models up to date. It is worth mentioning that the way the PSP framework is designed enables it to discard outlier events that may happen sporadically by some maintenance operations, backup invocation, or unplanned queries submission or data loads. This has been possible because 1) the MUM module uses a multi-layer historical data that does not count solely on the last data collected, 2) the Psychic adopts smoothing in its prediction models, and 3) the Shift Filtering mechanism of the Psychic factors out shifts (outliers) that will not positively contribute to the system performance.

The experiments described here accomplish two goals. The first goal is to assess the performance of the DBMS using the PSP framework. Experiments show that the prediction framework outperforms other normal modes of operation. The second goal is to demonstrate the adaptability of the framework. Our experiments show that the framework is robust against changes in the workload pattern. Although we experimented with a pattern transition that was swift and dramatic, PSP is able to learn the new pattern within the expected period of time. We consider the MUM to be a self-healing mechanism since it picks up the performance after its deterioration due to changes in the workload characteristics. We obtained similar results by experimenting with a number of scenarios generated by different scenario descriptors [17].

Our main contribution in this work is the PSP framework, which takes advantage of the low volatility and the cyclic patterns in a workload in order to allow a DBMS to follow proactive tuning strategies. The framework has the following strengths:

- It is efficient as it obviates the overhead caused by the expensive use of on-line prediction techniques that demand continuous monitoring for the system.
• It can estimate the best and worst performance under different modes of operations and, therefore, it can recommend the best mode suitable for a particular computing environment. Having prior knowledge about the expected performance helps in detecting performance violations. (e.g., a DBMS can alert DBAs by paging or emailing them if performance drops).

• It is generic as we speculate that this approach can be effective in other systems where their workloads exhibit some trend that makes them relatively predictable. Our methodology could be used to automate many other DBMS tasks such as determining when to make incremental backups, re-build indexes and refresh materialized views, update statistics, or reorganize data on the disk.

• The framework itself exhibits two important autonomic features. It is self-optimizing, as almost all of its internal parameters are determined automatically, and it is self-healing, as it adapts to new trends that may occur in the workload in order to retain the good performance.

• The use of our framework does not impose radical changes to the DBMS infrastructure, which promises a high degree of applicability to today’s large commercial DBMSs.

Possible future research directions include the following:

• Investigating the feasibility of using the Psychic-Skeptic framework to solve other types of database problems such as:
  
  • System Backup and Restore. Depending on the forecasted start and length of the idle period, the system may automatically perform incremental backup for specific portions of data.
  
  • Data Defragmentation and Reorganization. The system could anticipate the time periods at which it experiences low I/Os in order to rearrange data on disk in order to enhance their accessibility.
  
  • Updating Statistics. In DBMSs, query planning, optimization, and execution depend heavily on up to date statistics of the stored data. The system could exploit some idle periods to update these statistics.
  
  • Updating Indexes and Views. Auxiliary data structures such as indexes and materialized views can immensely improve system performance. However, maintaining these data structures can cause nontrivial overhead if performed at inappropriate times (e.g., at peak workloads). The system could predict under utilized periods of time and update its data structures during these periods.
  
  • Tuning the DBMS parameters as a function of the intensity of each workload type. Presently, DBMSs are tuned based on determining the dominant workload (e.g., either OLTP or DSS) using rule-of-thumb tuning strategies. An interesting research area would be to develop tuning strategies that take into account the intensity of each workload type (e.g., the DSSness degree) in the overall workload mix.
• Conducting an empirical study in which a feedback mechanism is established between the workload classifier and the DBA, which would allow the DBA to understand and correlate the currently observed performance with the workload type reported by the classifier. This would help the DBA develop better performance-tuning strategies. Furthermore, the feedback would allow DBAs to corroborate the workload type reported by the classifier and to determine if any retraining is necessary in order to improve the classifier’s prediction accuracy.

In conclusion, we trust that the workload put on the system is the most important input to it. Therefore, system engineers and designers should not solely focus on the optimization of the system’s architecture and its comprising components but they also should look for interesting characteristics in the workload that can be exploited to boost the performance of the system.

References


Trademarks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both: DB2, IBM, Intelligent Miner, Universal Database.

Microsoft and Windows are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.