TOWARDS WORKLOAD-AWARE DBMSS: IDENTIFYING WORKLOAD TYPE AND PREDICTING ITS CHANGE

by

SAID SELIM ELNAFFAR

A thesis submitted to the School of Computing

in conformity with the requirements for the degree of

Doctor of Philosophy

Queen's University

Kingston, Ontario, Canada

September, 2004

ABSTRACT

The type of the workload on a database management system (DBMS) is a key consideration in tuning its performance. Allocations for resources such as main memory can be very different depending on whether the workload type is Online Transaction Processing (OLTP) or Decision Support System (DSS). A DBMS also typically experiences changes in the type of workload it handles during its normal processing cycle. Database administrators must, therefore, recognize the significant shifts of workload type that demand reconfiguring the system in order to maintain acceptable levels of performance. We envision autonomous, self-tuning DBMSs that have the capability to manage their own performance by automatically *recognizing* the workload type and *predicting* its change over time.

In this thesis, we make two main contributions to the development of autonomic DBMSs. The first contribution is a methodology for automatically identifying a DBMS workload as either OLTP or DSS by building various classification models. We demonstrate the methodology with both industry standard workloads and with real workloads of global financial firms. The second contribution is a prediction architecture to forecast when the type of a workload may change. The DBMS can therefore proactively adjust its parameters, without incurring the overhead associated with the constant monitoring. We present experiments to show that the performance of the DBMS using our prediction mode outperforms other possible operation modes. They also show that the prediction architecture can adapt to changes in the workload pattern. The architecture does not demand human intervention and is potentially a generic solution for other similar prediction problems.

ACKNOWLEGMENTS

I would like to express my sincere appreciation to the many people who have supported me during my PhD journey. I am grateful to my supervisor, Dr. Pat Martin, who always believed in me and who has been there in good times and bad. I should thank our research associate, Wendy Powley, for her administrative/technical help, her moral support, and for sharing her office space and its associated funny atmosphere.

Special thanks go to my financial supporters: NSERC (Natural Sciences and Engineering Research Council of Canada), OGS (Ontario Graduate Scholarship), CITO (Communications and Information Technology Ontario), and IBM Canada where I spent many unforgettable, lovely days of my PhD internship. Thanks to the many people at IBM CAS (Centre for Advanced Studies), such as Berni Schiefer, Sam Lightstone, Kelly Lyons, and many others, with whom I was privileged to work.

I give a big hug to my cute daughters, Asalah and Sarah, who always reminded me that I need to take a break. And last but not least, to my wife, Niveen, who left heaven, as an angel, just to be beside me on Earth. She has been the best cure for any pain I went through. She has been always like the sunshine after the rain.

I am sure there are tens of others to thank. They know themselves, and they should know that I am grateful to them and they are all in my heart.

STATEMENT OF ORIGINALITY

I, Said Elnaffar, hereby certify that this PhD dissertation is original and all the ideas and inventions attributed to others have been properly referenced.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

1.1 MOTIVATION: THE NEED FOR AUTONOMIC SYSTEMS

The increasing power of computing systems and the desire to automate more tasks and processes means that systems are becoming too complex to manage and tune. A proposed approach, called autonomic computing, calls for systems that can manage and tune themselves automatically in order to reduce the total cost of their ownership. IBM is using the phrase "autonomic computing" to represent the vision of how IBM, the rest of the IT industry, academia, and the national laboratories can address this new challenge [43]. By choosing the word "autonomic," IBM makes an analogy with the autonomic nervous system. The autonomic nervous system frees our conscious brain from the burden of having to deal with vital but lower-level functions. Autonomic computing will free system administrators from many of today's routine management and operational tasks. Corporations will be able to devote more of their IT skills toward fulfilling the needs of their core businesses, instead of having to spend an increasing amount of time dealing with the complexity of computing systems.

The spiraling cost of managing complex computing systems is becoming a significant inhibitor that threatens to undermine the future growth and societal benefits of information technology. Simply stated, managing complex systems has grown too costly and prone to error. Administering a myriad of system management details is too labor-intensive. People under such pressure make mistakes, increasing the potential of system

outages with a concurrent impact on business. Testing and tuning complex systems is becoming more difficult. Consider the following:

- It is now estimated that one-third to one-half of a company's total IT budget is spent preventing or recovering from crashes [72][73].
- Nick Tabellion, CTO of Fujitsu Softek, said: "For every dollar to purchase storage, you spend $9 to have someone manage it." [29].
- Aberdeen Group studies show that administrative cost can account for 60 to 75 percent of the overall cost of database ownership (this includes administrative tools, installation, upgrade and deployment, training, administrator salaries, and service and support from database suppliers). [2]
- When you examine data on the root cause of computer system outages, you find that about 40 percent are caused by operator error [72], and the reason is not because operators are not well-trained or do not have the right capabilities. Rather, it is because the complexities of today's computer systems are too difficult to understand, and IT operators and managers are under pressure to make decisions about problems in seconds [9].

To respond, system design objectives must shift from the "pure" price/performance requirements to issues of robustness and manageability in the total cost of ownership equation. As a profession, we must strive to simplify and automate the management of systems. Today's systems must evolve to become much more self-managing, that is: self-configuring, self-healing, self-optimizing, and self-protecting.

1.2 AUTONOMIC COMPUTING

Automating the management of computing resources is not a new problem for computer scientists. For decades, system components and software have been evolving to deal with the increased complexity of system control, resource sharing, and operational management. Autonomic computing is just the next logical evolution of these past trends to address the increasingly complex and distributed computing environments of today. So why then is this something new? Why a call to arms to the industry for heightened focus and new approaches? The answer lies in the radical changes in the information technology environment in the few short years since the mid-1990s, with the use of the Internet and e-business extending environments to a dramatically larger scale, broader reach, and a more mission-critical fundamental requirement for business. In that time the norm for a large on-line system has escalated from applications such as networks consisting of tens of thousands of fixed-function automated teller machines connected over private networks to rich suites of financial services applications that can be accessed via a wide range of devices (personal computer, notebook, handheld device, smart phone, smart card, etc.) by tens of millions of people worldwide over the Internet. IBM's autonomic computing initiative has been outlined broadly. Paul Horn [43] described this "grand challenge" and called for collaboration toward developing autonomic computing systems that have characteristics as follows:

- To be autonomic, a system needs to "know itself"— and consist of components that also possess a system identity.
- An autonomic system never settles for the status quo—it always looks for ways to optimize its workings.

- An autonomic system must perform something akin to healing—it must be able to recover from routine and extraordinary events that might cause some parts to malfunction.
- A virtual world is no less dangerous than the physical one, so an autonomic computing system must be an expert in self-protection.
- An autonomic system cannot exist in a hermetic environment (and must adhere to open standards).
- An autonomic system must configure and reconfigure itself under varying and unpredictable conditions.
- An autonomic computing system knows its environment and the context surrounding its activity, and acts accordingly.
- Perhaps most critical for the user, an autonomic computing system must anticipate the optimized resources needed to meet a user's information needs while keeping its complexity hidden.

This thesis focuses on the last three characteristics. We believe that for a system to be autonomic, it should be aware of the properties of its workload, be able to anticipate their changes over time, and reconfigure itself accordingly. We apply these principles on one of the well-known complex computing systems, the Database Management System (DBMS), which is increasingly becoming part of almost every computing system. Realizing that expert Database Administrators (DBAs) are scarce and that they are a major part of the Total Cost of Ownership (TCO) makes an urgent call for an Autonomic DBMS (ADBMS).

1.3 THESIS HYPOTHESIS

We argue that for a complex system, such as a DBMS, to be autonomic it must know the important characteristics of its workload and how they change over time in order to be able to tune and reconfigure itself accordingly. The type of the workload presented to a database management system (DBMS) is a key consideration in tuning the system. Allocations for resources such as main memory can be very different depending on whether the workload type is Online Transaction Processing (OLTP) or Decision Support System (DSS). A DBMS also typically experiences changes in the type of workload it handles during its normal processing cycle. Database administrators must, therefore, recognize the significant shifts of workload type that demand reconfiguration in order to maintain acceptable levels of performance. We envision autonomous, self-tuning DBMSs that have the capability to manage their own performance by automatically recognizing the workload type and then reconfiguring their resources accordingly. In this thesis, we present an approach to automatically identifying a DBMS workload as either OLTP or DSS. We build a classification model based on the most significant workload characteristics that differentiate OLTP from DSS and then use the model to identify any change in the workload type.

Unfortunately, this methodology is still associated with run-time overhead primarily caused by the constant, on-line monitoring. Therefore, in this research, we present a prediction architecture that helps the DBMS forecast when the workload may change its type so the DBMS can proactively adjust its configuration parameters and resource allocations. Initially, the prediction system analyzes the historical data of a few days in order to construct a prediction model that

encompasses workload trends. Then it validates these trends using on-line models. The architecture is efficient as it outperforms other options that a DBMS could use to tap workload classifiers. Most importantly, it exhibits a high degree of adaptability as it can capture and learn any new pattern in the workload automatically.

1.4 RELATED WORK

To the best of our knowledge, there is no previous published work examining the problem of automatically identifying and predicting the type of a DBMS workload. There are, however, numerous studies characterizing database workloads based on different properties that can be exploited in tuning DBMSs [25]. Some studies use clustering to obtain classes of transactions grouped according to their consumption of system resources or according to the reference patterns in order to tune the system [94] or to balance the workload [67]. Some studies focus on characterizing the database access patterns to predict the buffer pool hit ratio [21] and the user access behavior [81]. Recent studies characterize DBMS workloads on different computer architectures in order to diagnose performance degradation problems [6] and to characterize the memory system behavior of the OLTP and DSS workloads [7]. Hsu et al. [44] systematically analyze the workload characteristics of TPC-C™ [89] and TPC-D™ [86] workloads, especially in relation to those of real production database workloads. This study shows that the production workloads exhibit a wide range of behavior, and in general, the two benchmarks complement each other in reflecting the characteristics of the production workloads.

1.5 CONTRIBUTIONS

The following points constitute the contributions spawned from this thesis:

- *An Automatic Workload Identification Methodology*. The methodology uses supervised machine learning techniques that analyze resource-oriented, low level characteristics of the workload. We note the following:
    - The methodology depends solely on the analysis of the system resource demands, which are easily obtainable from the typical system monitoring tools.
    - The methodology does not depend on any assumptions about the high level description of the SQL statements nor on any prior knowledge about the business or application domain.
    - The methodology does not merely identify the type of workload but also quantifies the intensity (e.g., DSSness and OLTPness percentages) of each type in the workload mix. Any subsequent performance tuning and modeling for the system control parameters should be a function of this intensity.
    - If there are more than two types of workloads, the same approach could be applied by increasing the number of class labels considered. Our experiments with the *GHC* classifier support this claim.
    - We believe that this independent approach (as it requires minimal human intervention and counts primarily on data obtained from the system itself) can be generally useful to other computing systems that need to automate the task of recognizing the type of the workload.

- This methodology has a high potential of being incorporated into today's commercial DBMSs[1]. One approach is to provide a set of prefabricated, ready-to-use workload classifiers for different popular workload types. A second approach is to adopt one of the hybrid classifiers that is trained on a wide variety of workloads.

- *Workload Prediction Architecture*. The Psychic-Skeptic architecture takes advantage of the low volatility and the cyclic patterns in the workload in order to allow the DBMS to follow proactive tuning strategies. The architecture has the following advantages:
  - 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 give an estimate of 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

[1] Due to its practical impact, this work yielded an IBM US/Canada patent pending.

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

- *Feasibility and Practicality of our Methods*. Interestingly, our solutions do not impose radical change to the DBMS infrastructure [18], promising a high degree of their practicality and applicability to today's large commercial DBMSs.
- *Progressing towards Autonomic Computing*. The workload identification and prediction methodologies presented in this thesis demonstrate how the DBMS can be workload-aware and more autonomic. In fact, our work fits in the *predictive* and *adaptive* levels of the revolutionary path towards having autonomous systems.
- *Surveying Workload Characterization Techniques and Methods*. As part of our research background, we surveyed a large number of case studies, across various computer disciplines, in order to identify the workload modeling techniques. In this study [25], we:
  - summarize the commonly used techniques for workload modeling,
  - describe the challenges that researchers typically face in characterizing workloads, and
  - propose a framework for workload characterization that serves as a guideline for constructing a workload model. In this framework we

suggest using data mining and data warehousing technologies in addition to the traditional analysis methods.

- *Implementing TPC-W Benchmark.* The TPC-W benchmark is among the other TPC benchmarks we used in this research due its interesting features. However, due to the lack of a suitable implementation, it was necessary for us to implement a TPC-W driver [88]. We trust that the implementation of this kit will be a useful addition to the research community[2].

1.6 ROAD MAP

The rest of this thesis is organized as follows. In *Chapter 2*, we explain the need for Autonomic DBMSs (ADBMSs) that are capable of managing and maintaining themselves. In this chapter, we examine the characteristics that a DBMS should possess in order to be considered autonomic. We assess the position of today's DBMSs by outlining example features from popular, commercial database products, such as DB2 UDB, SQL Server, and Oracle. We argue that today's DBMSs are still far from being autonomic. We highlight the source of difficulties towards achieving that goal, and sketch the most important research terrains that require investigation in order to have ADBMSs one day.

In *Chapter 3*, we present a survey of workload characterization techniques. Workload characterization is the process by which we produce models that are capable of describing and reproducing the behavior of a workload. Such models are imperative to any performance related studies such as capacity planning, workload balancing, performance

[2] A number of database research groups have experimented with our kit.

prediction and system tuning. In this chapter, we survey workload characterization techniques used for several types of computer systems. We identify significant issues and concerns encountered during the characterization process and propose an augmented methodology for workload characterization as a framework. We believe that the surveyed case studies, the described characterization techniques, and the proposed framework provide a good introduction to the topic, assist in exploring the different options of characterization tools that can be adopted, and provide general guidelines for deriving a good workload model suitable as an input to performance studies.

Chapter 4 discusses the workload identification problem. We start by stressing the role of the workload type with respect to tuning a DBMS by demonstrating how a number of configuration parameters can be differently set under OLTP and DSS workloads. Then we present our approach to automatically identifying a DBMS workload. We build a classification model based on the most significant workload characteristics that differentiate OLTP from DSS and use the model to identify any change in the workload type. We construct and compare classifiers built from two different sets of workloads, namely the TPC-C and TPC-H benchmarks and the Browsing and Ordering profiles from the TPC-W benchmark. We demonstrate the feasibility and success of these classifiers with TPC-generated workloads and with industry-supplied workloads.

In Chapter *5*, we present a prediction architecture that helps the DBMS forecast when the type of workload may change so that DBMS can proactively adjust its configuration parameters and resource allocations. In this chapter, we present this architecture and describe the functionality of its components. We show that the performance of the DBMS using our prediction mode outperforms other possible operation modes. We also show that it is adaptable to changes in the workload pattern, if any. Best of all, our prediction

methodology does not demand human intervention as almost all of its parameters are automatically estimated.

Finally, *Chapter 6* summarizes our work and sketches possible future research directions.

CHAPTER 2 AUTONOMIC DBMSS[3]

Database management systems (DBMSs) are a vital component of many mission-critical information systems and, as such, must provide high performance, high availability, excellent reliability and strong security. These DBMSs are managed by expert Database Administrators (DBAs) who must be knowledgeable in areas such as capacity planning, physical database design, systems tuning and systems management.

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

- **Increased emphasis on Quality of Service (QoS).** DBMSs are components of larger systems, such as electronic commerce applications, that support different levels of QoS depending on users' needs. 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. DBAs handle table reorganization, data statistics collection, backup control, security modeling and administration, disaster

[3] The work presented in this chapter is published [27] and co-authored by Powley, Benoit, and Martin.

recovery planning, configuration and performance tuning, problem analysis, and more.

- **Burgeoning database size**. Data warehouses containing tens of terabytes of data are not uncommon. Popular applications such as SAP typically create more than 20,000 tables and support thousands of users simultaneously [66].
- **E-Service era**. The problems described above become more apparent where the internet presents to the DBMSs a broad diversity of workloads with high variability under sophisticated multi-tier architectures.

DBMS customers and vendors, because of the spiraling complexity, have recently begun to place an increased emphasis on reducing the Total Cost of Ownership (TCO) of systems. Despite the dramatic recent growth in database sizes, TCO is increasingly dominated by human costs, specifically the DBAs. A 1998 study by the Aberdeen Group [1] showed an implementation of a leading industrial RDBMS incurred 81 percent of its TCO from the human costs of training, maintenance, and implementation. Similarly, a TCO report from D.H. Brown Associates [22] that compared two leading database products for both data warehouse and online transaction processing (OLTP) applications found that human costs represented a large component of TCO in all cases. Moreover, skilled DBAs and application developers are scarce.

Autonomic computing systems are a proposed approach to mitigate management complexity. In general, an *autonomic computing system* has the following properties [33]:

- The system is "aware of itself" and able to act accordingly.
- The system is able to configure and reconfigure itself under varying and unpredictable conditions.

- The system is able to recover from events that cause it to malfunction.
- The system is able to anticipate optimized resources needed to perform a task.
- The system is able to protect itself.

We believe that Autonomic Database Management Systems (ADBMS) are a desirable long-term research goal. In pursuing this goal it is useful to evaluate current DBMSs in light of the properties of autonomic computing systems in order to judge what has been accomplished to date and what problems remain to be solved.

This chapter has three goals. The first goal is to specifically define a set of features that a DBMS should have in order to be autonomic [27]. The second goal is to examine current DBMSs with respect to their embodiment of the concepts of autonomic computing systems [28]. We focus on three popular DBMS products, namely IBM DB2 Universal Database Version 8.1 [46], Oracle 9i [68] and Microsoft SQL Server 2000 [64]. Our objective is to report on the current state of practice with respect to autonomic DBMSs based on a review of generally available materials such as research papers, white papers and system documentation. We provide examples, not an exhaustive list, of autonomic features. We do not attempt to draw comparisons between the DBMSs. The third goal of this chapter is to highlight the present shortcomings and obstacles that hinder DBMSs from being autonomic.

In examining the DBMSs, we believe that the autonomic features available in the systems can be identified as belonging to one of the following general categories, which correspond to the kinds of tasks that are typically performed by DBAs:

- **Plug-n-Play DBMSs.** These features support system set-up and initialization. They include initial capacity planning, DBMS installation, configuration, and deployment, and data migration.
- **Physical and Logical Design.** These features include support for tasks related to laying out the data on the storage devices and structuring them properly. Examples of such tasks are the selection of the most efficient indexes and materialized views, and partitioning tables [79].
- **Ongoing Preventive Maintenance.** This category encompasses features that aim to keep the system stable and performing satisfactorily. They support the phase in which the DBMS monitors itself in order to perform ongoing tasks such as self-tuning and self-reorganizing. Examples include support for defragmenting data and re-structuring indexes, creating backups, updating statistics, space management, user management, and table and object maintenance.
- **Problem Diagnosis and Correction.** These features help with identifying any anomalies in the system and determining their root cause, notifying the administrators, taking corrective actions and tuning.
- **Availability and Disaster Recovery.** These features help the DBMS get back to its stable state or recover from a disaster. For example, the DBMS should be able to analyze its log carefully and identify the correct set of backup assets it retains in order to get the system operational. They also support multiple server synchronization and maintenance.

The remainder of the chapter is organized as follows. Section 2.1 presents our survey of the autonomic features of three popular commercial DBMSs. Section 2.2 summarizes the

survey and points out further functionality required in DBMSs to achieve the goal of autonomic DBMSs.

2.1 HOW AUTONOMIC ARE CURRENT DBMSs?

Ganek and Corbi identify important, general properties of an autonomic computing system [33]. In this section, we discuss DBMS-specific autonomic characteristics. We first detail what kind of automation a characteristic implies in the realm of DBMSs and then list some concrete examples drawn from commercial DBMSs that best match the description of the particular autonomic characteristic. We should note again that this is not meant to be an exhaustive list of features provided by the various DBMSs but instead we wish to outline where DBMSs are today in terms of autonomic capabilities.

2.1.1 Self-optimizing

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. Obviously, the most important task in need of optimization is the execution of a query. In fact, optimizing queries is one of the most apparent autonomic features of today's DBMSs. In general, query optimization involves query translation, the generation of a cost-efficient execution plan and dynamic runtime optimizations [34].

Producing accurate query plans depends heavily on statistics and column distributions. Oracle [71] and SQL Server [17] provide facilities that automatically determine which columns require histograms and also which tables require new statistics. Oracle also supports a dynamic sampling feature that gathers statistics on the fly.

In some cases, query optimization can be infeasible. Therefore, the DB2 optimizer allows the user to adjust the amount of optimization. More sophisticated models, such as those found in Oracle [71] and SQL Server [64], automatically determine the appropriate amount of optimization on a per-query basis. During query execution, cost models will be able to benefit from the self-validation of the cardinality model proposed by DB2's Learning Optimizer (LEO) [85].

Dynamic adjustment to the query execution strategy is one of the good features of present DBMSs. Oracle provides automatic memory allocation [71] so that each query has the appropriate amount of memory. DB2 and Oracle both provide an automatic query parallelism selection mechanism.

In addition to query optimization, a DBMS must also optimize the various utilities such as backup, restore, statistics collection and data load utilities. DB2's Load utility, for example, performs mass insertions of data into a target table by exploiting a series of parallel I/O sub-agents for pre-fetching, SMP parallelism degree, and the amount of memory available for buffering and sorting.

2.1.2 Self-configuring

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. An ADBMS should recognize changes in its environment that warrant re-configuration. It should also be able to reconfigure itself without severely disrupting online operations. A DBMS configuration includes performance parameters, resource consumption thresholds, and the

existence of auxiliary data structures such as indexes and materialized views in the database schema.

Typically, DBMSs provide configuration wizards such as DB2's Configuration Advisor. Configuration advisors are tools to assist with initial configuration but the settings are, in most cases, static. The goal of an autonomic DBMS is to provide dynamic adjustment of these settings. Little support is provided for this type of self-configuration. SQL Server and Oracle both provide some degree of automatic memory management. These systems allocate memory as needed by the database, limiting memory allocation when either a user-imposed limit is reached or the system's physical resources run low.

Self-configuring features of an ADBMS should include support for determining the optimal set of indexes and materialized views to be used by the query optimizer. All the DBMSs provide an index advisor (DB2's Design Advisor [59], SQL Server's Index Wizard [64], and Oracle's Index Tuning Wizard [68]) that recommends a suitable set of indexes. Similar to the index advisor, SQL Server [4] and Oracle [69] also recommend materialized views that can be beneficial to the system.

2.1.3 Self-healing

A fundamental requirement of a DBMS is that the database remains in, or can be restored to, a consistent state at all times. A DBMS must reliably log all operations, periodically archive the database and be able to use the logs and backups to recover from failure. Ideally an ADBMS should recognize when a full or incremental backup is necessary and perform these operations with minimal system disruption. In the event of catastrophic failure, an ADBMS should be able to retrieve the most recent backup, restore to the consistent point just before the failure, then resume its halted operations after handling the

exceptions. Oracle, for example, provides the ability to resume operations (such as a batch load) following corrective action (such as the addition of more disk space) [68].

All DBMSs support logging, backup and recovery mechanisms. DB2 has a recovery tool, the *Recovery Expert*, which 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. SQL Server and Oracle allow the DBA to set a recovery interval parameter that specifies a target for recovery time in seconds.

2.1.4 Self-protecting

Database protection implies at least the following aspects: database security [5], analytical auditing mechanisms, data encryption, and admission control strategies. These features shield the DBMS from potential, errant requests that may deteriorate its performance or bring the DBMS down.

All multi-user DBMSs provide authentication mechanisms that prevent unauthorized users from accessing the database. Database privacy ensures that users are granted access only to the portions of the database that are required. Current DBMSs differ in the level of access granularity; DB2 and SQL Server provide security on a per table basis whereas Oracle provides row-level security.

Admission and application control is essential for ADBMSs to protect the system from database requests that may deteriorate performance and/or undesirably consume system resources. The DB2 Query Patroller [46] and the Oracle Resource Manager [68] are examples of admission control tools used today.

2.1.5 Self-organizing

An ADBMS should be capable of dynamically re-organizing and re-structuring the layout of data stored in databases (e.g., tables), associated auxiliary data structures (e.g., indexes), and any system-related data (e.g., system catalog) in order to optimize performance. An ADBMS should assist in the initial layout of data on disks and should be able to shift data from one disk to another to even out disk demands. This ability is not present in current DBMSs, however Oracle does provide the ability to move tables while on-line [68].

To make efficient use of system resources, DB2, Oracle and SQL Server permit dynamic *online index reorganization* to reclaim leaf level storage. SQL Server has also the Partition Wizard and the Storage Design Wizard that help manage the layout of data cubes on disks.

2.1.6 Self-inspecting

Bowing to the principle *if you don't measure it then you don't know it*, an ADBMS should "know itself" in order to make intelligent decisions pertaining to all autonomic features discussed in the previous sections. The DBMS must collect, store and analyze relevant information about its components, performance, and workload. This information should be utilized in optimizing the performance, detecting any potential problems, updating statistics about the stored data, ensuring integrity of data read from disk, scheduling maintenance utilities, and in identifying interesting trends in the workload. The results of this constant inspection should be effectively presented to DBAs (using a GUI interface, for example) and be available as input for other autonomic components and operations.

Using the DB2 Health Center or the Oracle Manager Console, a DBA can examine the system for signs of unhealthiness and store performance data in a data warehouse. Such performance data can be further analyzed by analysis tools such as The DB2 *Performance Expert*.

The *Maintenance Advisor* is a tool that DBAs can use to examine DB2 statistics and make recommendations on what maintenance utilities should be run. Another example of automated inspection is DB2's ability to perform *Sector Consistency Checking* for page I/Os that ensures the integrity of read data by detecting any corruptions caused, for example, by incomplete I/Os.

2.2 ANALYSIS—WHAT IS MISSING?

Despite the many advances that have been made towards autonomic database management systems, much work remains to reduce the amount of human intervention required by these systems. We can summarize the most significant shortcomings in the following points:

- **High need for human input and intelligence**. Current DBMSs provide many tools and utilities to assist the DBA in tasks such as initial configuration, system monitoring and problem analysis, but in most cases these tasks still require a significant amount of input, intelligence and decision making on the part of the DBA.
- **Lack of Dynamic Adaptation**. Tuning advisors, for example, have proven useful in the initial setup of the database system, however, the settings, in most cases, do not automatically adapt to changes in the system environment or workload.

- **Lack of ability to reset DBMS parameters on-line.** Although close, DBMSs do not yet provide this capability. Note that being able to reset DBMS parameters dynamically is a mere prerequisite to enable autonomic features but it does not offer any kind of intelligent strategies.
- **Lack of analytical capabilities.** Many of the advisors and tools currently available are based on "rules of thumb" or heuristics that capture the human expertise programmatically. Robust analytical models and accurate prediction mechanisms are required for the more difficult tuning and configuration tasks.
- **No smart maintenance strategies.** Database utilities such as rebinding, statistics gathering, table and index reorganization and backup are currently provided by the DBMSs. However, an autonomic DBMS must have the ability to predict the best time to schedule the execution of these utilities.
- **Inability to run some operations on-line.** Some of the vital database operations such as deframenting data, updating statistics, and pruning important data structures like indexes can not be performed without bringing the DBMS down.
- **Lack of on-line schema evolution.** This feature should allow changing schema aspects without incurring an outage.
- **Lack of standard interfaces with other systems.** Current DBMSs do not show adequate enablement of autonomic features that allow smooth integration and synergy between the DBMS, as a middleware, and others components such as Web Servers.
- **No exploitation of characteristics of the workload.** Most of the current DBMSs overlook analyzing the characteristics of the workload and its behavior over time.

- **Trivial security and privacy strategies.** Current security and privacy features do not offer any kind of clever strategies that help the DBMS develop or change its protective plans. For example, ADBMSs should provide *auditing* mechanisms where logs are used to track all DBMS activity. The DBMS can use this information to track trends, analyze potential threats, support future security planning, and assess the effectiveness of countermeasures. Agrawal et al. propose more interesting ideas to improve DBMSs' privacy [5].

Despite the efforts undertaken by industry-led projects such as IBM's SMART and Microsoft's AutoAdmin, we have not witnessed a real change to the DBMS infrastructure that is necessary, as a rigorous but flexible architecture, for making the transition to a fully autonomic system.

2.3 SUMMARY

Autonomic DBMSs, that is DBMSs that can manage themselves, are an attractive solution to complexity and total cost of ownership problems associated with DBMSs. We examine three popular database products, namely DB2, Oracle and SQL Server, with respect to their autonomic computing features. We find that, while all three products now contain features of an ADBMS, there is still a long way to go before we can claim that DBMSs are autonomic computing systems.

We conclude that ADBMS research should focus in four main areas. The first area is the development of a proper infrastructure to allow the clean introduction of autonomic computing system features. Current research literature proposes two very different paths to ADBMSs. One is a revolutionary approach that argues for a complete redesign of

DBMSs with fine-grained components [61][37] or components that provide a RISC-like interface [93]. This RISC-style facilitates individual management of the components and controlled interactions between them. The second approach is evolutionary [33] and identifies a set of phases that existing systems can be taken through in order to become autonomic systems. We feel the evolutionary approach is more realistic for existing DBMS products.

The second main area of research for ADBMSs is intelligent decision-making tools. It is important that DBMSs become able to independently analyze and act upon the information they collect about their performance. A key component of this progress will be the development of effective mathematical models and feedback control loops that can be used to make more accurate performance prediction and reach better tuning decisions.

The third main area of research is the development of a useful model of the system itself. A model must exist in order for DBMSs to know themselves. The model will have to represent the resources used by the ADBMS efficiently, the relationships between these resources, the workload of the ADBMS and the current state of the ADBMS.

The fourth main area of research is making DBMSs more intelligent by discovering knowledge from data surrounding them, especially from their workloads. An ADBMS will require workload characterization techniques [25] to automatically extract the necessary information from this data. Statistical models and data mining techniques [24][26] can help tap interesting properties in the DBMS's workload. This research direction is the ultimate motivation of our thesis as we present methodologies that allow a DBMS to automatically recognize the workload type and efficiently predict its variation over time.

Finally, we do not think that progressing towards ADBMSs will mean the demise of DBAs. It will mean the end of repetitive administrative tasks, freeing DBAs to spend more time on new applications and on the business policies and strategies. Furthermore, DBAs will be needed to evaluate and select recommendations before they are implemented. Once comfortable with system recommendations, DBAs can enable a DBMS to take actions automatically and simply report on them.

CHAPTER 3 WORKLOAD CHARACTERIZATION

The performance evaluation of computer systems requires understanding of a system's workload. As shown in Figure 1, the workload is a set of requests, or components, that place different demands on various system resources. Workload characterization provides a model of a system's workload by means of quantitive parameters and functions. The model should be representative, compact, and accurate [10], and should be able to describe and reproduce the dynamic behavior of the workload and its most essential static features.

We explore various techniques other researchers use, in different computer fields, in order to extract workload characteristics. This study helps us choose the appropriate characterization methods to identify important properties of the DBMS workload (Chapter *4*) and model its behavioral change over time (Chapter *5*).

The chapter has three goals[4]. First, we summarize the most common techniques used to characterize workload, such as graph-based techniques, stochastic processes, clustering, and numerical fitting. We give a brief description of these techniques and classify them according to their ability to extract different aspects of the workload, that is, the static properties or the dynamic behavior. Second, we organize these techniques within a common framework. To this end, we present a general methodology for workload characterization that is used in our research. Our third goal is to point out the potential problems and concerns that may be encountered during the characterization process.

---

[4] A detailed discussion of the material presented in this chapter can be found in [25].

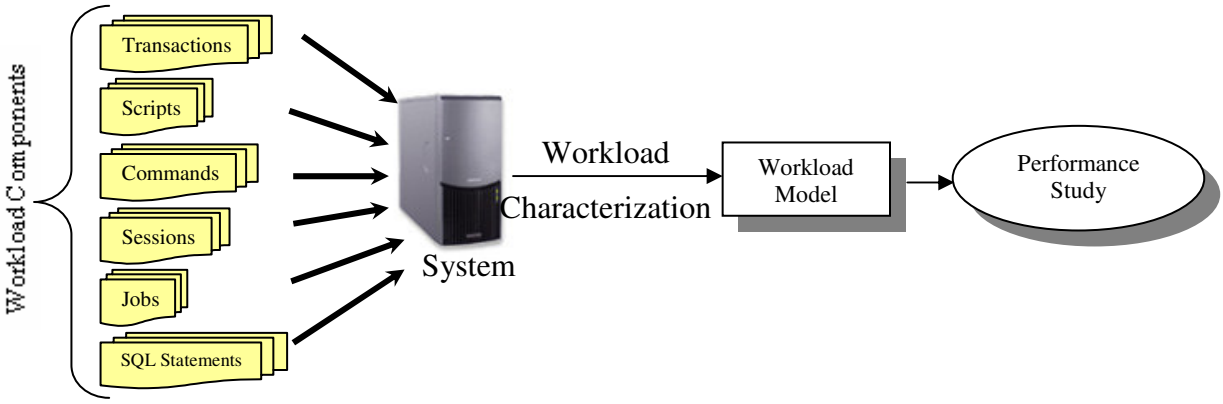


Figure 1. The workload characterization process

The rest of the chapter is organized as follows. Section 3.1 gives a brief description of the common techniques used in workload characterization. Section 3.2 summarizes workload characterization techniques found in different types of computer systems, namely batch and interactive systems, client/server systems, databases systems, parallel systems, and WWW systems. Section 3.3 explains our workload characterization framework and highlights the most significant concerns and potential problems that researchers encounter during the characterization process.

3.1 CHARACTERIZATION TECHNIQUES

In this section, we briefly describe the techniques most commonly used to analyze system workloads. The selection of a particular technique depends mainly on the purpose of the performance study, and on the level of detail required. It might be necessary, in some cases, to evaluate more than one technique in order to select the best one.

Functionally, we can classify the characterization techniques into two main categories: *static* and *dynamic*. Static techniques explore the intrinsic characteristics of the workload, such as transaction classes, the correlation between workload parameters and component dispersion, which do not change over time. Examples of these techniques are clustering,

principal component analysis, averaging, and correlation coefficients. Dynamic techniques, such as Markov models, user behavior graphs, and regression methods, focus on describing the behavior of the workload and the way it fluctuates over time. These techniques usually analyze the historical data of the workload and, as a result, aid in forecasting its behavior in the future.

Throughout the workload characterization process, adopting one technique is usually not sufficient to obtain a complete analysis; several techniques may be used in combination in order to come up with an approach that satisfies the research needs. For example, clustering techniques might be used to classify the transactions submitted to the system. Afterwards, each class may become a node in User Behavior Graphs [15], or a transitional state in a Markov model. This example raises another issue, namely the importance of obtaining both static and dynamic properties of the workload in order to obtain a complete picture.

Visualization tools, such as graphs, histograms, and fitting curves, are a key means of highlighting significant features in the workload under investigation while simple techniques, like averages, may smooth out some details such as *burstiness*. Sections 3.1.1 and 3.1.2 describe static and dynamic characterization techniques respectively. Table 1 summarizes the techniques examined in these sections.

Technique Type	*Technique*	*Advantages*	*Disadvantages*
Static	Descriptive Statistics (average, variance/standard deviation, correlations, distributions)	o Provides preliminary description o Easy to calculate	o May not be sufficient; further analysis is needed
	Single-parameter Histogram	o Expressive visual means o Shows frequencies of each bin o Frequency distribution can be used in simulation models	o Incapable of expressing the correlation among different parameters
	Multi-parameter Histogram	o Illustrates the correlation between different parameters o Expressive visual means	o Difficult to plot the correlation between more than two parameters
	Factor Analysis (e.g., Principal Component Analysis)	o Simplifies performance data and reduces their dimensionality	o Complex to calculate
	Clustering	o Identifies homogeneous classes of workload components based on certain criteria	o Difficult to choose the appropriate number of clusters
Dynamic	Markov Models (Markov chains, Markov processes)	o Predicts the order in which the requests are executed	o Complex to calculate
	Prediction Using Neural Networks	o Performs short-term and long-term forecasting of workload parameter values	o Difficult to design and to configure
	Moving Average	o Useful for short-term, single value prediction o Easy to calculate	o Cannot perform long-term forecasting o Cannot predict more than one single value o No special consideration for the most recent observations o Difficult to determine the best number of observations
	Exponential Smoothing	o Useful for short-term, single-value forecasting o Places more weight on the most recent observations o Easy to calculate	o Cannot perform long-term forecasting o Cannot predict more than one single value o Difficult to determine the best smoothing weight
	Regression Methods (linear and non-linear fitting)	o Predicts the value of a parameter as a function of others o Identifies trends	o Can be complex to calculate
	User Behavior Graphs	o Used mostly in interactive systems o Describes the user's probable transition to a particular command/transaction type	o Requires clustering to compose the nodes
	Probabilistic Attributed Context Free Grammar	o Used in hierarchical systems (e.g., client/server) o Translates views of higher layers to lower layers	o Cannot be used to map lower layers to higher ones

Table 1. Static and dynamic workload characterization techniques.

3.1.1 Static Techniques

Static techniques, such as descriptive statistics, single-parameter histogram, multi-parameter histograms, principal component analysis, and clustering, help explore the static characteristics of the workload. In this section we give a brief description of each type.

Descriptive Statistics. Descriptive statistical techniques are used to identify the static properties of the workload. Using these techniques helps describe what the workload parameters look like: where their center (average) is, how broadly they are spread (dispersion or variance), and how they are related to each other (correlation).

Averaging, or *arithmetic mean*, is the simplest method to characterize a workload parameter such as user think time, number of active users, number of I/O operations required to execute a query, or inter-arrival time of transactions. Averaging presents a single number that summarizes the parameter values observed. However, it is not always appropriate to count on arithmetic mean; the *median*, *mode*, *geometric mean*, or *harmonic mean* should be used in some cases.

The average alone is not adequate if the performance data has high variability. Variability is usually specified by the *variance*. However, the *standard deviation*, which is the square root of the variance, is more useful in expressing the variability because it has the same unit as the mean. The ratio of the standard deviation to the mean is called the *coefficient of variance* (C.O.V). A zero C.O.V. indicates that the measured parameter is constant. In this case, the mean gives the same information as the complete set. A high C.O.V. indicates high variance, in which case it may be useful to look at the complete histogram (discussed below). There are also other alternatives for specifying variability

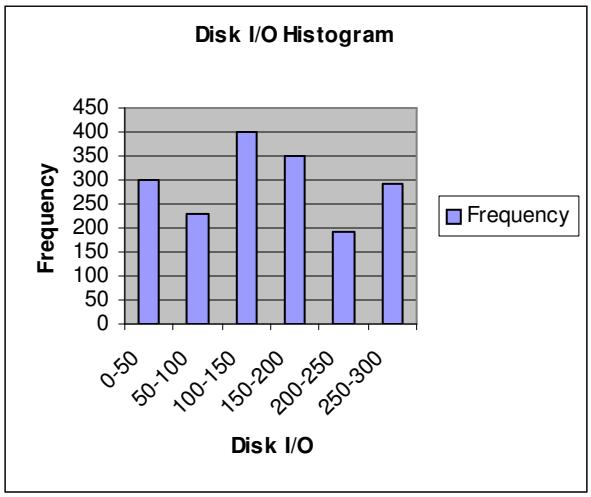


Figure 2. A simple histogram of disk accesses

like *range* (minimum and maximum), *10th*- and *90th*- *percentile*, *semi-interquartile range*, and the *mean absolute deviation*.

Correlation is another useful statistical technique to discover the relationship between different workload parameters. It is a decimal fraction, called *correlation coefficient*, which indicates the degree to which the parameters are related. There are numerous ways (e.g., Biserial, Point Biserial, Tetrachoric, Spearman rank-order, etc.) to calculate the coefficient of correlation. *Pearsonian product moment*, commonly called *Pearsonian r*, is the most popular one [83].

Single-parameter Histograms. A histogram is a visual representation of a parameter where the range of values is divided into intervals called *bins*. As shown in Figure 2, the histogram displays the frequency of the observations of each bin. This frequency distribution is used in simulation models to generate a test workload. However, one of the

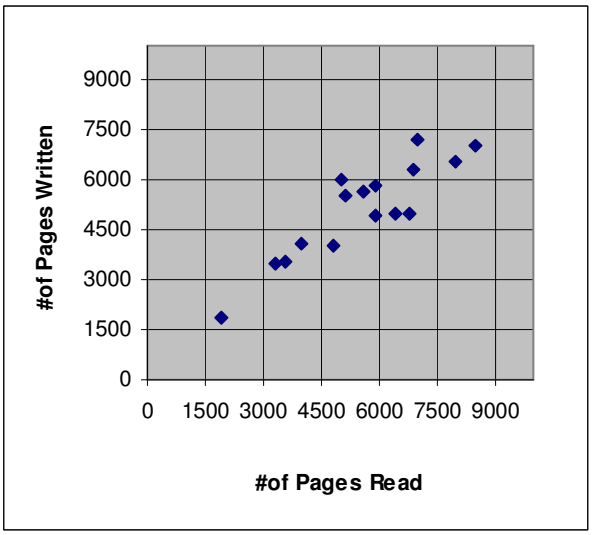


Figure 3. A two-parameter histogram

drawbacks of a histogram is that it is incapable of expressing the correlation among different parameters. Therefore, multi-parameter histograms can be used instead.

Multi-parameter Histograms. Multi-parameter histograms illustrate the correlation between different workload parameters. The distribution of *n* workload parameters can be described by an *n*-dimensional matrix or histogram. Figure 3 shows an example of a two-parameter histogram that represents the number of read and written pages in a database system. Each dot in the figure represents a system node. The number of dots in a cell of the grid represents the number of nodes that read and wrote pages in the range corresponding to the cell. As can be seen, the nodes reading a large number of pages are also the ones that write a large number of pages. Therefore, a significant correlation may exist between the two parameters. On the other hand, we should note that it is difficult to plot multi-parameter histograms that correlate more than two parameters.

Factor Analysis. The term *factor analysis* usually refers to statistical techniques that describe multidimensional sets of data by means of geometric representation. Their goal

is to help choose a subspace of the variable space such that the projection of the data set on that subspace preserves as much information of the original set as possible. Consequently, factor analysis is beneficial for simplifying data and reducing their dimensionality.

Principal Component Analysis (PCA) [39] is a factor analysis technique that maps a set of parameters, or variables, into another set, called principal components, characterized by orthogonality among the components and by linear dependence on the parameters in the original set. PCA is an iterative process in which the first component is chosen such that it maximizes the variance of the linear function expressing the dependence of the transformed parameters on the original ones. The second component is chosen such that it maximizes the remaining variance while this component must be orthogonal to the first, and so on.

Clustering. Clustering is one of the most widely adopted techniques in workload characterization [15][74][67][23]. Clustering identifies homogeneous groups, or classes, of workload components, based on the similarity of resource demands. In general, clustering methods can be classified as hierarchical or non-hierarchical. Hierarchical techniques, like the *Minimal Spanning Tree (MST)* [80] method, start by assuming that each component of a workload is a cluster. Then, the two clusters with the minimum distance are merged to form a single cluster. The process iteratively continues until either all the workload components are grouped into a single cluster or the desired number of clusters is reached. On the other hand, the non-hierarchical techniques, like the *k-means* algorithm [40], start from an initial partition that consists of the exact desired number of clusters. Workload components are reassigned among clusters so that a particular cluster criterion, known as *distance function*, is optimized.

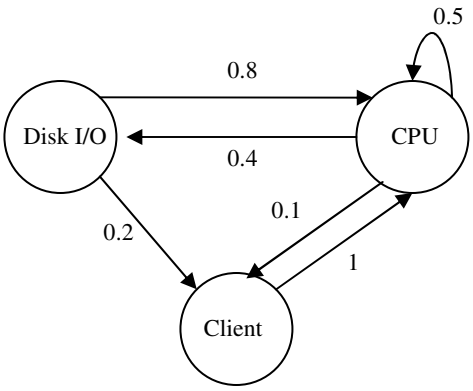


Figure 4. A state transition diagram representing a Markov model.

Deciding on the number of clusters is a common problem in any cluster analysis study. Generally, it depends on the goal of the study and it is desirable to keep this number small for practicality. Various clustering algorithms are available in the literature [51].

3.1.2 Dynamic Techniques

Next, we examine techniques commonly used to describe and predict the behavior of the dynamic aspects of the workload.

Markov Models. Knowing the number of requests of each type, or class, is not sufficient. It is also important to know the order in which requests are executed in the system. If it is assumed that the next request depends only on the current one, then the requests follow a *Markov model* [38]. This model can be represented by a *transition matrix*, which gives the probability of moving to the next state given the current one. A corresponding *state transition diagram* can be easily constructed from the transition matrix. Figure 4 shows an example of a transition diagram in which the probability of a job using the disk after visiting the CPU is 0.4, the probability of it returning to the CPU from the disk is 0.8, and so on.

Markov models are used to describe the transitions between any system states, not just between system resources. For example, in a software development environment that provides several types of software tools, we can use a transition matrix to describe the probability of transitions between the different types of development tools like editors, compilers, linkers, and debuggers.

Prediction Using Neural Networks. Although getting a perfect prediction is a very hard problem, neural networks can be used to obtain reasonably good predictions in some cases [62]. *Feedforward* as well as *recurrent* networks are commonly used for this purpose. The prediction problem can be viewed as a function approximation problem, in which the function values are represented as *time series*, that is, a sequence of values measured over time. Based on the knowledge of the most recent d values of a time series, the neural network can be trained to predict the d+1 future value. The accuracy of predicting the values of a parameter may increase if a multivariate time series and the correlations among all workload parameters are taken into account [63].

Typically, two types of predictions are considered: short-term, or *one-lag,* and long-term, or *multi-lag*, predictions. In one-lag predictions, the forecasting of the future value is based just on the past actual values. Multi-lag prediction also exploits some of the predicted values in order to predict future values. An example of multi-lag prediction is forecasting the value of a time series a year from today while the values for the next eleven months are unknown.

Moving Average. This is a simple prediction technique in which the next forecasted value is the average of the previous ones. This method shows very good results if the data are almost stationary, that is, with little variation [56]. However, it is not suitable for

long-term prediction as it is not capable of predicting more than a single value at a time. The forecasted value can be calculated as follows:

$$f_{t+1} = \frac{x_t + x_{t-1} + \ldots + x_{t-n+1}}{n}$$

where f_{t+1} is the forecast value for period t+1, x_t is the actual value at time t, and n is the number of previous observations. It is not always easy to determine the number of periods, n, that should be used. Thus, different values of n may be examined in order to find the one that achieves the least *mean squared error* (MSE), which is calculated as follows:

$$MSE = \frac{\sum_{t=1}^{n} (x_t - f_t)^2}{n}$$

Exponential Smoothing. Exponential smoothing is similar to the moving average in terms of using the average to predict the next value. It is particularly useful for short-term forecasting and when the data are stationary. However, it differs from the moving average in the way it calculates the forecast value; it puts more weight on the most recent historical observations. The idea stems from the hypothesis that the latest observations give a better indication of the future. Here, the forecast value f_{t+1} is calculated as follows:

$$f_{t+1} = f_t + \alpha(x_t - f_t)$$

where α is the smoothing weight (0<α<1). Again, some values of α are better than others in terms of getting the least MSE, and additional tests help to choose a suitable one.

Regression Methods. The value of a variable, called the *dependent* variable, can be predicted as a function of other variables, called *independent* variables, using regression models. Many mathematical forms exist, which describe the relationship between these

variables. A linear relationship is a common assumption used to estimate the values of the dependent variable [63].

User Behavior Graphs. User Behavior Graphs (UBG) are considered as the basis for several workload models [12]. They are similar to the state transition diagrams used in Markov models and are commonly used to describe the workload of interactive systems, such that each user has her own UBG [30]. A UBG is a probabilistic graph whose nodes represent the different command types issued by the user, and whose arcs represent the transition from one command type to another throughout a user session.

Probabilistic Attributed Context Free Grammar. A Probabilistic Attributed Context Free Grammar (PACFG) [32] is a central means of constructing generative workload models, especially in systems that have a hierarchical nature, like client/server and WWW environments [53][78]. A PACFG can translate views between the different layers of the system hierarchy. For example, a PACFG can map the client-oriented view of the workload, such as commands submitted during user sessions, to a low-level system view like TCP/IP protocol requests.

A PACFG is a 3-tuple $G_A = \{G, A, Q\}$ where G is the regular grammar defined as $G = \{V_N, V_T, P, S\}$. V_N and V_T are a set of non-terminal and terminal symbols, respectively, P represents a set of production rules, S is the start symbol, A is a set of attributes and Q is a set of probabilities associated with P. At each layer in the hierarchy, the system supports a set of operations that are represented by non-terminals. The mapping of a particular layer's operations to the operations of the next layer is achieved by expanding each of the non-terminals to a sequence of non-terminals or terminals at the next lower level. Such an expansion is controlled by the production rules (P) and the associated set of probabilities (Q). Each non-terminal has two attributes s and e, which respectively denote the start and

end times of an operation, such as a user session, occurring at a particular layer. The duration of an operation is the difference between *s* and *e*.

3.2 CASE STUDIES

We now present a summary of case studies[5] of different types of computer systems, namely batch and interactive systems, client/server systems, database management systems, parallel systems, and WWW systems. Throughout these case studies, we focus on the workload characterization aspects of each system type, and identify the most commonly used techniques. We summarize the features of the commonly used techniques, in each computer field, in a table. A row in a table represents a major characterization technique (e.g., clustering) used to analyze the workload of that type of computer system. The columns of the table describe the technique used, the workload properties explored (i.e., static or dynamic), the methods used (e.g., *k*-means), the approach followed (i.e., functional or resource-oriented), the basic workload component considered (e.g., transaction, URL, session, etc.), the input parameters analyzed (e.g., # of files), the workload type (i.e., interactive, batch, or scientific), the purpose of the study, some of the useful results obtained, other techniques used in combination with this technique, and some references to case studies that used this technique.

3.2.1 Batch and Interactive Systems

A number of characterization techniques appeared in early studies of interactive and batch computer systems (e.g., [12] and [14]). Interestingly, these techniques are still the

[5] Detailed in a technical report [25].

basis of approaches adopted in recent studies of various computer systems. In general, the different clustering and factor analysis techniques are commonly used to describe the static aspects of the workload while stochastic processes, numerical fitting techniques and graph based approaches are used to capture the dynamic behavior of the workload as it changes over time. Table 2 summarizes the commonly used characterization techniques in batch and interactive systems.

Technique	Static/ Dynamic	Method	Approach	Component	Parameters	Workload Type	Purpose	Results	Other Techn. Combined
Clustering	Static	*k*-means, Minimal Spanning Tree	Resource-oriented (Mostly), Functional	Job, program	CPU time, #of files, #of Job steps, #of I/Os per device type	Batch, Interactive	Distinguish program/comm and/task classes according to their functionality or resource consumption	(7-9) homogeneous groups	Markov Models, User Behavior Graphs
Factor Analysis	Static	Principal Component Analysis	Resource-oriented	Program	CPU time, disk I/Os, language, memory	Batch	Reduce dimensionality of Data; Identify characteristics of each program types	(2-3) factors accounting for the majority of the variance in the data set	Clustering
Prediction Models	Dynamic	Markov Chain	Functional	Job	Task type, user state, timestamp	Interactive	Describe user/system behavior as it transits from one state to another	Transition matrix or state diagram describing the probable sequence of jobs or tasks	Clustering
Numerical Fitting	Dynamic	Linear and non-linear Regression	Resource-oriented	Job	Arrival rate, timestamp	Interactive	Model workload arrival pattern	Parametric model of high-degree polynomial functions	Clustering
Graphs	Dynamic	User Behavior Graph	Functional, resource-oriented	Command	Command type and sequence, resource consumption	Interactive	Describe user/system behavior as it transits from one state to another	A probabilistic graph model	Clustering

Table 2. Characterization techniques used in batch and interactive systems.

3.2.2 Client/Server Systems

A client/server system consists of clients connected to servers via a network. Distributed file systems, distributed database systems, distributed multimedia systems, and WWW applications, are examples of client/server systems. They can be seen as a hierarchical structure composed of three layers: client, network, and server. Characterization techniques and tools may differ depending on the layer in which the characterization is taking place (e.g., [19][20]). Table 3 summarizes the commonly used characterization techniques in client/server systems.

Technique	Static/ Dynamic	Method	Approach	Component	Parameters	Workload Type	Purpose	Results	Other Techn. Combined
Statistics	Static, dynamic	Exponential/ Poisson processes, Bimodal distributions, variance, averages, time-domain analysis based on R/S statistic	Functional – characterizing load per protocol or file type	File, packet	#of requests, inter-arrival time, file references, #of I/Os, packet lengths, error rate	Interactive	Model the reference behavior at file servers	Distributions of requests and their arrival; discovering self-similarity and burstiness; understand user behavior	-
Graphs	Dynamic	User Behavior Graph (UBG)	Functional, resource-oriented	Command	Arrival time, command type, message length, source/destination addresses	Interactive	Model the workload at user, system, network levels (separately)	A model representing user and system behavior	Clustering
Grammar	Dynamic	Probabilistic Attributed Context Free Grammar (PACFG)	Functional	Session, command, network request	Arrival time, command type, network request type	Interactive	Build a generative model that considers the hierarchical nature of client/server	A model used as input to a simulation model	Clustering
Clustering	Static	*k*-means	Functional, resource oriented	User sessions, commands, network requests	Command type, network request type	Interactive	Reduce the analyzed data	Distinctive groups of commands and network requests	PACFG, UBG

Table 3. Characterization techniques used in client/server systems.

3.2.3 Database Management Systems

As database management systems (DBMS) become increasingly popular and are often part of larger systems, they require considerable tuning to get them working at an optimal performance level [58]. As in any computing system, identifying the characteristics of the workload should aid in tuning and configuring these systems more effectively (see for example, [44][81]). Table 4 summarizes the commonly used characterization techniques in database systems.

Technique	Static/ Dynamic	Method	Approach	Component	Parameters	Workload Type	Purpose	Results	Other Techn. Combine
Clustering	Static	Heuristic, Neural network, *k*-means	Resource-oriented	DB transaction	#of database calls, #of locks, #of references	Batch, interactive	Capacity Planning, load balancing	(4-8) clusters	Statistics
Prediction Models	Dynamic	Markov chain	Functional	DB Query	Think time, sequence of executing queries in a session	Interactive	Predict buffer hit ratio; make predictive prefetching; enhance caching	A predictive model for the near future executed queries	Statistics
Numerical Fitting	Dynamic	Non-linear regression (e.g., Levenberg-Marquardt method)	Functional	DB transaction	Arrival time	Interactive	Model the arrival pattern of transactions	Degree 8 of exponential polynomial	Statistics
Statistics	Static, dynamic	Basic statistics summaries (avg., distributions), non-homogeneous Poisson process, histograms	Functional	DB transaction, application, SQL statement	Arrival time, cache miss rate, memory footprint, # of references to a memory block, read/write page accesses, CPU demands, deadlocks, number of transactions completed, #of different pages accessed per transaction	Interactive	Understand memory behavior in different architectures; enhance buffer replacement and concurrency control	Degree 8 of exponential polynomial, cache reuse distribution, identifying locality/sequentiality of reference.	Numerical fitting

Table 4. Characterization techniques used in database systems.

3.2.4 Parallel Systems

Parallel applications are developed to solve problems in a shorter time and/or to solve larger problems in the same time. To meet these objectives, we need to tune, debug and diagnose the performance, which requires characterizing the parallel applications [13]. Essentially, the characterization requires collecting measurements by adding instrumentation to the source code of the application, to the operating system scheduler, or to the communication libraries [42]. Table 5 summarizes the commonly used characterization techniques in parallel systems.

Technique	Static/ Dynamic	Method	Approach	Component	Parameters	Workload Type	Purpose	Results	Other Techn. Combined
Graph	Dynamic	Profiles, shapes, phases	Resource-oriented	Application	# of processors being busy computing or communicating at certain time	Scientific	Identify application phases	Graphs of communication, computation, and I/O phases	Statistics
Statistics	Static	Signatures, avg. computation/ communication time, avg. # of sent/received messages, avg. message length, # of I/O operations	Resource-oriented	Application	Timing parameters: execution, computation, communication times, I/O times, Volume parameters: # of communications, I/O operations, floating-point operations	Scientific	Obtain inherent characteristics of a parallel application run on a specific number of processors.	Single-value metrics, and signatures	-

Table 5. Characterization techniques used in parallel systems.

3.2.5 World Wide Web Systems

Internet-based systems can be classified as client/server systems described in Section 3.2.2, but we opt to discuss them separately because the literature is rich with research papers pertaining to workload analysis issues (e.g., [77][95][36]. This should not be a surprise as we observe, day after day, the explosive increase of Internet popularity. Table 6 summarizes the commonly used characterization techniques in the World Wide Web systems.

Technique	Static/ Dynamic	Method	Approach	Component	Parameters	Workload Type	Purpose	Results	Other Techn. Combined
Analytical Modeling	Static, Dynamic	Queuing Network Model, Layered Queuing Model, Method of Layers	Functional, resource-oriented	URL	CPU time, disk I/Os, Response time, URL sequences, URL type	Interactive	Study QoS measures for E-Commerce Server	The mean response time is a good indicator for the 90th percentile of response times	Statistics
Prediction Models	Dynamic	Hybrid LRS-Markov models	Functional	URL	URL sequences	Interactive	Predict the navigation behavior of WWW surfers	Producing simple and accurate model better than Markov's	Markov Model
Statistics	Static	Averages, distributions, histograms, correlations, COV, Hurst parameter	Functional	URL, File	URL timestamps, Transfer size, file size, file type, request success rate, client locality	Interactive	Discover Web invariants	Discovering ten web invariants	-
Grammar	Dynamic	Probabilistic Attributed Context Free Grammar	Hierarchical -Functional, resource-oriented	User session, HTTP request, TCP/IP request	URL time stamps, URL type, think time	Interactive	Profile requests submitted to WWW servers	Map the high user-oriented view to low TCP/IP requests	-
Data mining and data warehousing	Static, Dynamic	OLAP: Drill-down, roll-up, slice-and-dice; DM tools: association, clustering, classification, transition/trend analysis	Functional, Resource-oriented	URL	Timestamps, IP addresses, transfer size, URL type, file size	Interactive	Discover interesting characteristics in Web log	Discovering web access patterns and trends	Prediction techniques and time-series analysis, Clustering, Statistics
Graph	Dynamic	Customer Behavior Model Graph (CBMG)	Functional, Resource-oriented	Session	Timestamps, #of hits, # of pages viewed, #of visits	Interactive	Model E-commerce workloads	Deriving useful metrics from CBMG like avg. # of visits per user state, avg. session length, buy to visit ratio	Clustering

Table 6. Characterization techniques used in World Wide Web systems.

3.3 CHARACTERIZATION FRAMEWORK

In this section, we propose a general framework for the workload characterization process. The framework makes the process more systematic, emphasizes some essential steps needed to derive a good workload model, and prevents some common problems. A correct characterization process does not have to strictly follow all the steps recommended by the framework. However, in our research, the framework helps us keep the big picture of the whole process in mind so we do not miss any essential requirement.

The main difficulties that may be encountered throughout the workload characterization process are:

- **Difficulty of System Instrumentation**. Systems need to be instrumented in order to obtain performance measurements. This may require the insertion of some probes, like counters, into the system itself or into the operating system. This task is challenging due to the complexity of the systems and the typical absence of the source code.
- **System Disturbance**. Instrumenting the system is an intrusion that adds extra overhead. Hence, the degree of intrusion should be minimized to reduce the perturbation of the system's behavior under the investigated workload.
- **Complexity of Analyzing Large Volume of Performance Data**. A large amount of system measurements are needed to construct a workload model [13], which increases the complexity of managing and analyzing the data.
- **Validating Model Representativeness**. Assessing the workload model representativeness, that is, how accurately the model represents the real workload, is a key issue [63]. Normally, modeling tends to hide some details that might be desirable

to study. Hence, a careful decision should be made about the model's abstraction level in the *requirements analysis phase* (explained next). This should help identify how much information loss can be tolerated and what important features must be included in the model.

- **Model Compactness**. The characterization process should result in a compact model. It is impractical for the workload model to incorporate all the basic components of the real workload. Ideally, a compact workload model should place a much smaller demand on the system than the actual workload [63].

Ferrari et al.[31] describe a methodology for constructing a workload model. We augment their methodology to produce a framework that introduces the following additional concepts:

- *Creating a Performance Database*. Building a database for the workload parameter values provides a robust way of storing and managing large volume of performance data. It also provides a solid foundation for the application of any analytical technique that might be adopted in the subsequent phases.
- *Distinguishing between the static and dynamic techniques*. This distinction is sometimes important in the analytical phase in order to choose the appropriate tool, and to create an adequately descriptive executable workload model.
- *Using data warehousing and data mining technologies*. In addition to the traditional analytical and statistical techniques commonly used in workload characterization, we suggest in this framework exploiting the capabilities of the data warehouse technology [16] and data mining tools.

The *multi-dimensional data cube* in a data warehouse provides operations such as *drill-down*, *roll-up*, and *slicing and dicing*. These operations offer online analytical processing (OLAP) capabilities, including an engine for deriving various statistics, and a highly interactive and powerful data retrieval and analysis environment. The data warehouse approach also overcomes the complexity problem stemming from processing large data sets.

Besides the OLAP tools, the analytical capacity can be extended further by adopting data mining techniques, which can discover implicit knowledge in the performance data that can be expressed in terms of rules, charts, tables, graphs, and other visual forms for characterizing, classifying, comparing, associating, or predicting the workload. Data mining techniques have been used to discover interesting patterns and features in customers' data that may lead to better marketing strategies. Similarly, in the workload characterization framework, we mine for interesting patterns and key characteristics in the system's workload. The integrated use of data warehousing and data mining has proven useful in analyzing web logs [95] and we believe that using both technologies as part of the workload characterization methodology would be beneficial too. Figure 5 shows the framework of the workload characterization process. Deriving a workload model consists of three phases: requirements analysis phase, construction phase, and validation phase. Next, we describe these phases and explain the tasks involved in each of them.

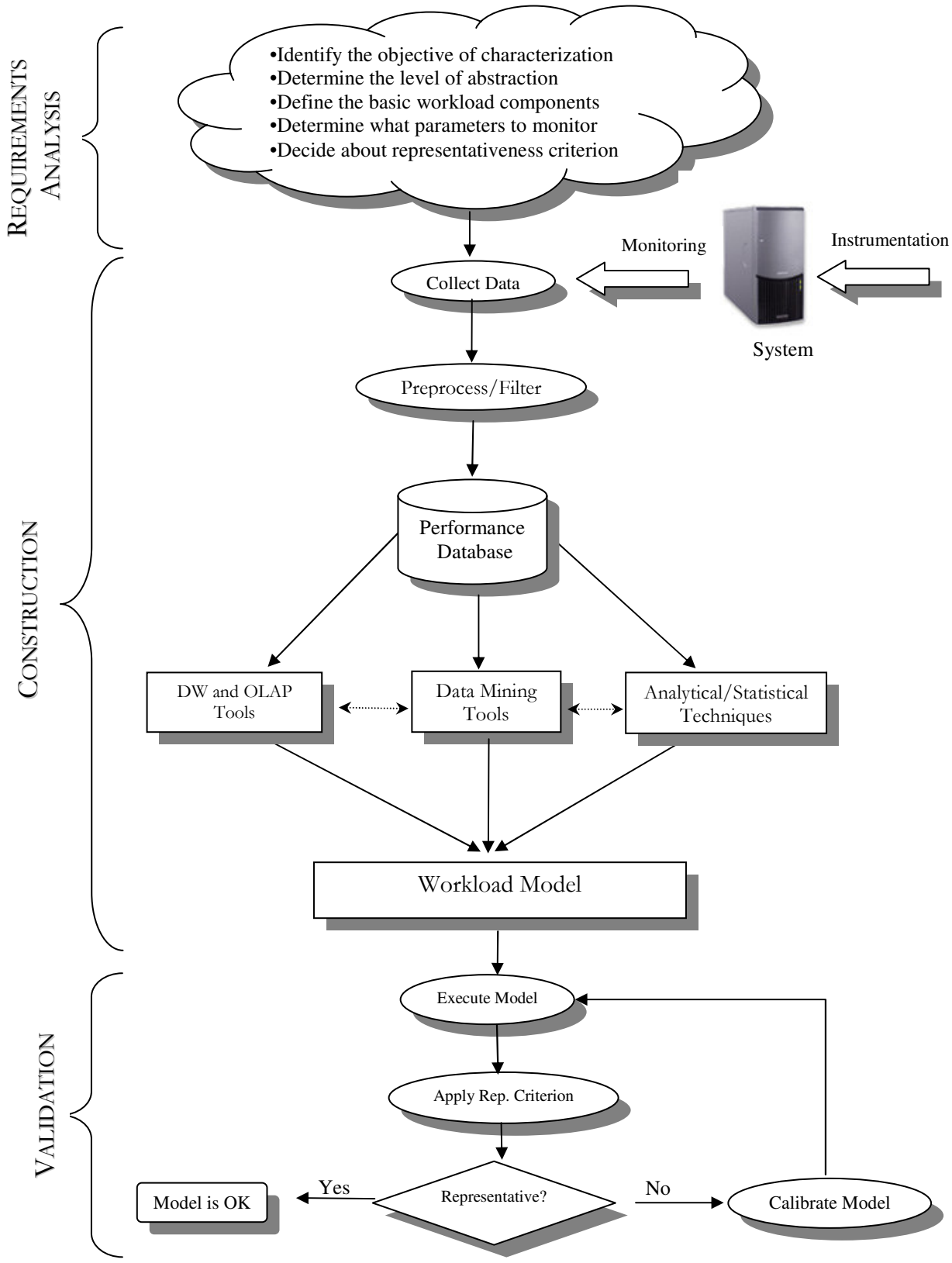


Figure 5. Workload Characterization Methodology.

3.3.1 Requirements Analysis Phase

The reasons for characterizing a system's workload should be clear from the beginning because they help derive the appropriate workload model. Therefore, based on a clear identification of the goals of the performance study, analysts must determine the following:

Abstraction Level. Depending on the intended use of the model, the level of abstraction at which the characterization will take place should be determined. The system can be viewed as a hierarchy; the highest level in this hierarchy is *functional* and the lowest one is *physical*. At the functional level, for example, the analyst may focus on identifying the types of applications executed in the system, identifying the kinds of web objects that are requested frequently, or grouping database transactions according to their functionality. At the physical level, they may categorize workload components, such as transactions, TCP/IP requests, or user interactive commands, according to their resource consumption (e.g., CPU, I/O, and memory). The higher the level, the lower the amount of detail with which the workload can be described. The selection of the level of detail helps in making other decisions like the choice of the *basic workload component*.

Basic Workload Component. The smallest unit of work must be determined. As shown in Figure 1, a workload component can be an application, a script, a command, a SQL statement, a user session, a transaction, a CPU instruction, a request, or a job. For example, applications and CPU instructions can be considered as basic workload components at the functional and physical levels, respectively.

Workload Parameters. Depending on the abstraction level and the basic workload component, parameters are chosen to give a quantitive description of the workload

components. Examples of workload parameters are packet size, arrival time, number of I/O instructions, memory space demand, and number of file handles required. It is preferable to choose parameters that are dependent on the workload rather than on the system. For example, response time and CPU time are not appropriate as workload parameters since they are highly dependent on the system currently executing the workload. The ratio of the read-only requests to the update requests in a database workload is a good candidate parameter as it effectively describes the workload under study and has nothing to do with system configuration. In particular, those characteristics that have an impact on the system performance should be included in the workload parameters. Parameter selection may also be restricted by the capability of the monitoring tools currently available in the system.

Criteria of Evaluating Model Representativeness. The criteria of evaluating the accuracy and representativeness of the derived model should be determined. They are used to validate the model as explained in Section 3.3.3.

3.3.2 Model construction Phase

This phase consists of the following three main tasks:

Collecting and Preprocessing Performance Data. During the measurement interval, the workload parameter values are collected from the system. The raw data may not be ready for direct analysis, so, further processing may be needed to put the data in a clean state and in an appropriate format. For example, the raw data set usually contains noise and outliers that may distort the results of the subsequence analysis. Furthermore, some type of transformations might be needed in this step. For example, if one of the parameter's density functions is highly positively skewed, a logarithmic transformation is needed.

Creating A Performance Database. After preprocessing and filtering the raw data, a relational database is created to store the performance data. The database facilitates information extraction and data summarization based on individual attributes.

Analysis Stage. Analyzing the workload parameter values aims to extract the workload's static and dynamic features. In Section 3.1, we described some of the tools commonly used to perform the static and dynamic analyses. The static analysis tools explore the intrinsic features of the workload and partition the workload components into homogeneous classes or groups. However, in order to make the derived workload model executable we need to capture the characteristics of the workload over time in order to reproduce the correct workload mixes. Hence, the dynamic properties of some time series are considered. Stochastic processes, numerical fitting techniques, and the various predictive models are useful in describing the behavior of the workload over time. As depicted in Figure 5, the traditional analytical/statistical techniques and the proposed data mining and OLAP tools can be used, separately or together, to analyze the performance data in order to characterize the workload.

Analyzing the static characteristics helps to choose representative components (mixes) that can reflect the key properties of the real workload. Analyzing the dynamic behavior of the workload completes the picture by describing the distribution and the sequence of execution of these workload components. Determining the static and dynamic characteristics of the workload can be the ultimate goal of the workload characterization because such knowledge can be adequate to facilitate tuning and enhancing the system's performance. Hence, the characterization process may stop at this point. However, the model can be further processed to generate an executable, runnable model that can be practically ported to different systems to assess their performance. A benchmark is an

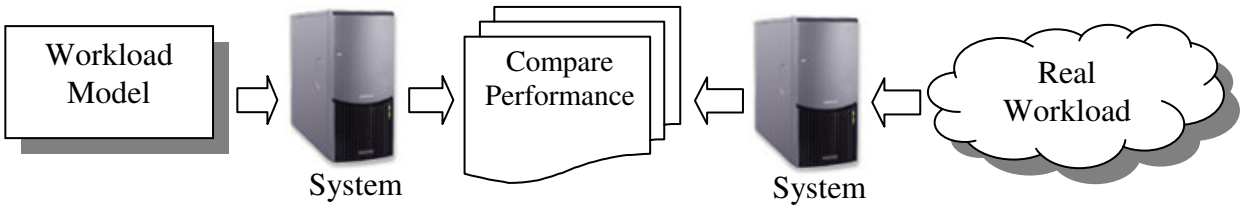


Figure 6. Validating the representativeness of a workload model

example of an executable workload model. The executable format of the workload model is also a means of its validation, as explained next.

3.3.3 Model Validation Phase

Validating the workload model is not always straightforward. One way of examining the accuracy of the derived workload model is to assess its effect on the system compared with the effect of real workload [31]. As can be seen in Figure 6, if the performance measurements resulting from the application of the workload model and the real workload are the same or proportional, then we have a good model. For example, Keeton and Patterson [52] proposed and evaluated simplified *microbenchmarks* for studying the architectural behavior of database workloads. These microbenchmarks pose simple queries of the database to generate the same dominant I/O patterns exhibited in more complex, fully-scaled workloads like TPC-C and TPC-D. One of the potential benefits from this microbenchmark approach is smaller hardware requirements. The representativeness of the new models is evaluated by comparing the processor and memory system characteristics of the microbenchmarks with that of fully scaled workloads running on similar hardware. These metrics were selected because most fully scaled database servers are configured with enough disks to be CPU bound; hence

processor and memory behavior are important factors in determining database performance [7].

Other techniques of validation may take into account criteria like arrival time of components and the resource usage profile [50]. If the derived workload model does not provide sufficient accuracy then some calibration of its parameters (static characteristics) or for its component mixes (dynamic characteristics) is required. The calibration process is repeated until a satisfactory level of representation is reached.

3.4 SUMMARY

Characterizing the system's workload is an essential early step in any performance study. Although workload characterization, like performance evaluation, is still considered more of an art than a science, the methodology discussed in this chapter can be deemed a general framework for deriving a workload model. A substantial amount of details in this framework are highly dependent on the objectives of the performance study as well as on the type of system. We propose using data warehousing and data mining technologies as a promising analytical approach. It may provide a potential solution for some of the well-known problems in workload characterization like the difficulty of managing large volumes of performance data sets and the complexity of analyzing them. This should lead to better scalability, more interactivity, and a variety of different, possible analyses.

The wide range of analytical techniques discussed in this chapter can be used to extract the static and dynamic characteristics of the workload. More than one technique may be combined in order to obtain the desired model. In general, we have noticed that identifying distinct classes in the workload using the various clustering techniques is the main goal of many studies.

The notion of multi-layer workload characterization has been adopted by many workload characterization studies. It is based on viewing the system as a hierarchical structure, which allows the characterization process to take place at any level in this hierarchy. For example, in network-based systems, characterization can be accomplished at many levels: user level, application level, protocol level, and network level.

A multi-layer characterization allows insight into how changes at the upper levels can affect the lower levels, and enables the prediction of the impact of new applications or systems. By analyzing the measures collected from each layer, a model of the overall workload of the system can be obtained. Nonetheless, we have found that most of the studies characterize the workload of each layer separately. Probabilistic graphs techniques, such as User Behavior Graphs, have been commonly used for modeling the workload at each layer.

Workload characterization typically relies on analyzing performance data collected from the system. The choice of what to measure depends on the objective of the study, the workload features to be evaluated, the level of abstraction (or details) required, and the availability of monitoring tools to collect the proper measures. The selection of what to measure is critical. Indeed, there is a tradeoff between the amount of detail to be measured and the perturbations caused by monitoring. Measurements collected from the system are not only important to the analysis phase; they are also useful for parameterizing the derived models with empirical data drawn from the real system. In some cases, such parameterization is essential to obtaining a successful model.

However, and as already pointed out, obtaining the *proper* measurements from the system is sometimes challenging. For example, web logs have been used as the primary source of system data to model the workload of WWW applications. While this may

reflect the actual use of the resources on a site, it does not record reader behaviors like frequent backtracking or frequent reloading of the same resource if the resource is cached by the browser or a proxy. Other means of data gathering like client-site log files collected by the browser, or a Java Applet have been suggested. However, while these techniques solve such problems, they demand the user's collaboration, which is not always available. In some systems, for example networks, special equipment such as network cards, bridges, routers, and gateways, constituting the network-based systems make the characterization process much harder. As a result, new measuring tools have been devised in order to collect parameter values from the system.

We believe that workload characterization will remain the focus of researchers and will constantly keep progressing in order to exploit newly introduced techniques and to cope with the requirements of new computer architectures. We also believe that no matter what new performance-oriented architectures have to offer toward enhancing performance, the notion of characterizing the workload and identifying its features should always lead to better improvements.

CHAPTER 4 WORKLOAD IDENTIFICATION

Database administrators (DBAs) tune a database management system (DBMS) based on their knowledge of the system and its workload. The type of the workload, specifically whether it is Online Transactional Processing (OLTP) or Decision Support System (DSS), is a key criterion for tuning [49][70]. For example, in an OLTP environment, which is characterized by a high degree of concurrency, it is imperative to have a sufficient number of database agents in order to process the workload efficiently. In a DSS environment, intra-parallelism is indispensable for processing the very complex, long queries, whereas it is undesirable to OLTP workloads. A change in the workload type is a reflection of a change of users' tendency of utilizing a particular class of applications over others. In addition, a DBMS experiences changes in the type of workload it handles during its normal processing cycle. For example, a bank may experience an OLTP-like workload by executing the traditional daily transactions for most of the month, while in the last few days of the month, the workload becomes more DSS-like due to the generation of financial reports and running long executive summaries. Unless we automate the process, DBAs must recognize the significant shifts in the workload and reconfigure the system accordingly in order to maintain acceptable levels of performance. Experts use rule-of-thumb tuning strategies to handle each workload. We provide a limited discussion of such tuning strategies (Section 4.1) as they are beyond the scope of this work.

This chapter is structured as follows. Section 4.1 presents the motivation behind identifying the workload type and spells out its significance with respect to configuring a DBMS. Section 4.2 introduces the workload identification problem. Section 4.3 explains

the approach and the methodology we use to solve this problem. Section 4.4 presents several sets of experiments that validate our approach.

4.1 OLTP VS. DSS: WHAT DIFFERENCE DOES IT MAKE?

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. Next, we illustrate how to set some of these parameters in light of a given workload type using experts' rule-of-thumb recommendations. These recommendations are derived from the technical documentation of DB2 [46] and from our empirical

experience. We trust that such recommendations are effective for other major DBMSs such as SQL Server and Oracle.

4.1.1 Different Configuration for Different Workloads

There are a number of DBMS tuning parameters that are set differently for each workload type. Examples of these parameters are:

MAXAGENTS. This parameter indicates the maximum number of DBMS agents. Having sufficient number of agents to process the workload is imperative especially in an OLTP workload where multiple concurrent users (or connections) access the database.

CATALOGCACHE. This parameter indicates the maximum amount of space that the catalog cache can use from the database heap. Sufficient memory allocated to this parameter helps preparing execution strategies for SQL statements. This cache is used to obtain information about the definition of the database, tablespaces, tables, indexes, and views. If all the required information is available in the cache, the DBMS can avoid disk I/Os and shorten plan preparation times. Having a high cache hit ratio (95 percent or better) is key for OLTP applications.

LOCKTIMEOUT. This parameter specifies the number of seconds that an application will wait to obtain a lock. This helps avoid global deadlocks for applications. For an OLTP workload, LOCKTIMEOUT should be set to a low value (e.g., 5-10 seconds). If a transaction cannot get the locks it needs to complete its work then it should timeout quickly and allow other transactions to execute. For a DSS workload, LOCKTIMEOUT should be set to larger value (e.g., 60 seconds).

MINCOMMIT. This parameter allows the DBMS to delay the writing of log records to disk until a minimum number of commits have been performed. This delay can help

reduce the overhead associated with writing log records and as a result improve performance when you have multiple applications running and many commits are requested within a very short time frame. MINCOMMIT is a powerful tuning parameter for OLTP workloads that process high volumes of transactions per second. When properly set, I/Os to logs are grouped together — resulting in fewer log I/Os. In a DSS workload, MINCOMMIT can be safely set to 1, whereas it should be set to a higher value in OLTP environments.

LOGBUFSZ. This parameter specifies the amount of the memory to use as a buffer for log records before writing these records to disk. This logging buffer should be larger in OLTP databases than in DSS.

INTRA_PARALLEL. This parameter specifies whether the database manager can use intra-partition parallelism. This parameter should be set to NO for OLTP workloads and YES for DSS workloads.

MAX_QUERYDEGREE. If INTRA_PARALLEL is enabled, this parameter specifies the maximum degree of intra-partition parallelism that is used for any SQL statement. An SQL statement will not use more than this number of parallel operations. Since parallelism is generally undesirable for OLTP workloads, it is recommended to set MAX_QUERYDEGREE to 1. A proactive, cautious DBA would set MAX_QUERYDEGREE to a value equal to the number of CPUs on the system in case the DSS workload is dominant.

SHEAPTHRES & SORTHEAP. The SHEAPTHRES parameter works in concert with the SORTHEAP parameter to govern sort memory. SHEAPTHRES dictates a soft limit for total sort memory used by all databases, and SORTHEAP controls the limit on memory for any one sort. OLTP workloads usually perform few sorts, and the sorts are typically

small. A small value of SORTHEAP (e.g., 128) is sufficient for OLTP database. That value results in double the number of concurrent sorts without increasing the SHEAPTHRES memory.

DSS workloads perform many possibly large sorts. SORTHEAP memory is also used for hash joins. For a DSS, at a minimum, the SHEAPTHRES should be doubled or tripled (e.g., between 40,000 and 60,000) and the SORTHEAP size should be sufficiently large (e.g., between 4,096 and 8,192).

DFT_QUERYOPT. This parameter is used to direct the optimizer to use different degrees of optimization when compiling SQL queries. Since an OLTP transaction is typically short, relatively simple, and properly indexed, selecting an access plan should not take much DBMS computing power. Therefore DFT_QUERYOPT should be set low so that the DBMS spends minimal time preparing its access plan. Spending a few seconds *thinking* about the SQL access strategy when actual execution only takes a quarter of a second is a waste.

On the other hand, SQL in a DSS query is very complex and often consumes large quantities of CPU and I/O resources, so a few extra seconds of time spent preparing the SQL access plan can be a wise investment, especially if it yields an access strategy that trims minutes or hours off of a query's elapsed time. In a DSS environment, DFT_QUERYOPT should be set high.

CHNGPGS_THRESH. Asynchronous page cleaners write changed pages from the buffer pool to disk before the space in the buffer pool is required by a database agent. This means that the agents do not have to wait for a changed page to be written out, before being able to read a page, and an application's transactions should run faster. The

CHNGPGS_THRESH parameter specifies the level (percentage) of changed pages at which the asynchronous page cleaners will be started, if they are not currently active.

The default value for CHNGPGS_THRESH is 60 percent, meaning that when 60 percent of the pages in the buffer pools become dirty, then the asynchronous I/O cleaners begin writing the changed pages out to disk. For a DSS workload, the default usually delivers good results. For a high-transaction volume OLTP workload, lowering the CHNGPGS_THRESH from 60 percent to 50 or 40 percent can be beneficial.

NUM_IO_CLEANERS. This parameter specifies the number of asynchronous page cleaners that write changed pages from the buffer pool to disk before the space in the buffer pool is required by a database agent. Performance experience indicates that asynchronous write I/Os are usually at least twice as fast as synchronous write I/Os, so it is important to try to achieve an asynchronous write percentage (AWP) of 90 or higher. In an OLTP environment, it is recommended that the number of NUM_IO_CLEANERS is incremented until either 90 percent of writes are performed asynchronously or the number of NUM_IO_CLEANERS is equal to the number of CPUs. If the latter limit is reached, we consider making gradual reductions in CHNGPGS_THRESH until AWP is greater than 90, but set CHNGPGS_THRESH no lower than 30 percent.

In a DSS environment, DB2 uses NUM_IO_CLEANERS for writing to TEMPSPACE, temporary intermediate tables, index creations, and more. Therefore, we recommend setting NUM_IO_CLEANERS equal to the number of CPUs.

NUM_IO_SERVERS. This parameter specifies the number of I/O servers used to prefetch data into the DBMS's buffer pools. To achieve maximum parallelism in a DSS workload, it imperative to have enough prefetchers available. However, most OLTP applications do not require prefetching.

4.2 WORKLOAD IDENTIFICATION PROBLEM: INTRODUCTION

Workload characterization is important in a world that increasingly deploys "universal" database servers that are capable of operating on a variety of structured, semistructured and unstructured data and across varied workloads ranging from OLTP through DSS. Universal database servers, such as IBM® DB2® Universal Database™ [49], allow organizations to develop database skills on a single technology base that covers the broad needs of their business. Universal databases are increasingly used for varying workloads whose characteristics change over time in a cyclical way. Most of the leading database servers today fall into this category of universal database, intended for use across a broad set of data and applications.

The goal of our research is to develop a methodology by which a DBMS can automatically identify the workload type. This is an important step towards autonomic DBMSs, which know themselves and the context surrounding their activities, and can automatically tune themselves to efficiently process the workloads put on them [33]. This problem is challenging for a number of reasons:

- There are no rigorous, formal definitions of what makes a workload DSS or OLTP. We currently have only general rules such as:
    - Complex queries are more prevalent in DSS workloads than in OLTP workloads.
    - A DSS workload has fewer concurrent users accessing the system than does an OLTP workload.

- An autonomous computing solution requires that the DBMS identify the workload type using only information available from the DBMS itself or from the operating system (OS); no human intervention is involved.
- A solution must be online and inexpensive. This entails adopting lightweight monitoring and analysis tools to reduce system perturbation and minimize performance degradation.
- A solution must be tolerant of changes in the system settings and in the DBMS configuration parameters.
- A solution should assess the degree to which a workload is DSS or OLTP, that is, the concentration of each type in the mix. Any subsequent performance tuning procedure should be a function of these degrees.

Our solution treats workload type identification as a data mining classification problem, in which *DSS* and *OLTP* are the class labels, and the data objects classified are database performance *snapshots*. We first construct a workload model, or a *workload classifier*, by training the classification algorithm on sample OLTP and DSS workloads. We then use the workload classifier to identify snapshot samples drawn from unknown workload mixes. The classifier scores the snapshots by tagging them by one of the class labels, DSS or OLTP. The number of DSS- and OLTP-tagged snapshots reflects the concentration (relative proportions) of each type in the mix.

We validate our approach experimentally with workloads generated from Transaction Processing Performance Council (TPC) benchmarks[6] and with real workloads provided by three major global banking firms. These workloads are run on DB2® Universal Database™ Version 7.2 [49]. We construct and evaluate two classifiers. One classifier, which we call *Classifier(C, H)*, is built using OLTP and DSS training data from the TPC-C™ [89] and TPC-H™ [90] benchmarks, respectively. As an OLTP system benchmark, TPC-C simulates a complete environment where a population of terminal operators executes transactions against a database. The benchmark is centered around the principal activities (transactions) of an order-entry environment. These transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses. While the benchmark portrays the activity of a wholesale supplier, TPC-C is not limited to the activity of any particular business segment, but, rather, represents any industry that must manage, sell, or distribute a product or service. On the other hand, TPC-H is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance. This benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions.

The second classifier, which we call *Classifier(O, B)*, is built using OLTP and DSS training data from the Ordering and Browsing profiles of the TPC-W™ [91] benchmark,

[6] Note that since our TPC benchmark setups have not been audited per TPC specifications, our benchmark workloads should only be referred to as TPC-like workloads. When the terms TPC-C, TPC-H, and TPC-W are used to refer to our benchmark workload, it should be taken to mean TPC-C-, TPC-H-, and TPC-W-like, respectively.

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.

Results obtained from testing the genericness of these classifiers show that every workload is a mix of its own set of SQL statements with their own characteristics and properties. Therefore, very specialized classifiers such as *Classifier(C, H)* and *Classifier(O, B)* are not expected to always be successful. Nevertheless, we believe that we can construct a generic classifier that is able to recognize a wide range of workloads by combining the knowledge derived from the analysis of different flavors of DSS and OLTP training sets. Such a generic classifier could be incorporated into a DBMS to assist in tuning the system.

We present two generic classifiers. The first one, the *hybrid classifier (HC)*, is constructed by training it on a mix of the TPC-H and the Browsing profile workloads as a DSS sample, and a mix of the TPC-C and the Ordering profile workloads as an OLTP sample. The second generic classifier, the *graduated-hybrid classifier (GHC)*, considers the TPC-H (Heavy DSS or *HD*) and the Browsing profile (Light DSS or *LH*) as different intensities or shades of DSS workloads, and the TPC-C (Heavy OLTP, or *HO*) and the

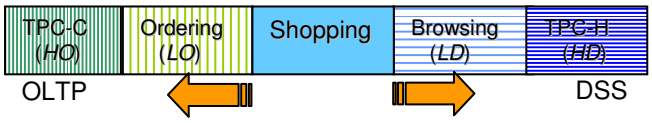


Figure 7. Different Shades of DSS and OLTP

Ordering profile (Light OLTP, or *LO*) as different shades of OLTP workloads in its recognition (Figure 7). In other words, *GHC* attempts to qualitatively analyze the different aspects of the DSS and OLTP elements in the workload by reporting the concentration of each workload shade comprising each type. Besides having practical advantages, with respect to adopting finer tuning strategies that suit different shades of each workload type, *GHC* demonstrates that our approach can be applied to workloads of more than two types.

The remainder of this chapter is organized as follows. Section 4.3 describes our approach to the problem and the selection criteria for the snapshot attributes. We then outline our methodology and how to compose the snapshots. Section 4.4 describes the sets of experiments with the four classifiers, and discusses the results obtained from experimenting with the benchmark and industry-supplied workloads.

4.3 APPROACH

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. The workload itself contains valuable information about its characteristics that can be extracted and analyzed using data mining tools. Our approach is to therefore use data mining classification techniques, specifically Decision Trees Induction [65], 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.

4.3.1 Overview

Classification is a two-step process. In the first step, we build a model (or *classifier*) to describe a predetermined set of data classes. The model is constructed by analyzing a training set of data objects. Each object is described by attributes, including a *class label attribute* that identifies the class of the object. 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 the second step, we use the model for classification. First, the predictive accuracy of the classifier is estimated using a test data set. If the accuracy is considered acceptable (for example, reporting that 80%, or more, of the tested snapshots are classified as DSS or OLTP when we attempt to identify a DSS -or OLTP-deemed workload), the model can be used to classify other sets of data objects for which the class label is unknown.

For our problem, we define the DSS and OLTP workload types to be the two predefined data class labels. The data objects used to build the classifier are performance snapshots taken during the execution of a training database workload. Each snapshot reflects the workload behavior (or characteristics) at some time during the execution and is labeled as being either OLTP or DSS. We tried building our classifiers using two methods. We used the SPRINT [84], a fast scalable decision-tree based algorithm, and the neural network (NN) classification using feed-forward network architecture and the back-propagation

learning algorithm. Both algorithms are implemented in IBM® DB2® Intelligent Miner™ Version 6.1 [48], which we used to design and configure our classifiers.

We found that the decision tree classification method is better for our problem than the neural network method for several reasons. First, the decision tree method, as expected, is easier to use and to set up than the neural networks method. Second, it is easier to interpret and explain the results from the decision tree method. Third, the decision tree method provides the ability to assign weights to the attributes that reflect the importance of the attributes to the decision process. Finally, the decision tree method achieved a higher accuracy in tests than the neural network algorithm. Table 9 describes the settings we used in the decision tree algorithm that we adopted in implementing our methodology.

4.3.2 Snapshot Attributes

The data objects needed to build the classifier are performance snapshots taken during the execution of the database workload. Each snapshot reflects the workload behavior at some time during the execution. We use the following criteria to select the snapshot attributes:

1. *Relevance*. Select attributes that play a role in distinguishing between DSS and OLTP mixes.
2. *Accessibility from the System*. Select attributes that are readily and inexpensively obtainable from the DBMS or the operating system at run time.
3. *Low System-Dependence.* Select attributes that are less sensitive to changes in the system settings or to DBMS configuration parameter changes. System settings include operating system resource allocations, such as memory and CPUs, and the

database schema. DBMS configuration parameters include buffer pool sizes, sort heap size, isolation level, and the number of locks.

We first considered the following list of candidate attributes for the workload snapshots:

1. Queries Ratio(%): The ratio of SELECT statements versus Update/Insert/Delete (UID) statements, which is usually higher in DSS than OLTP.
2. Pages Read: DSS transactions usually access larger portions of the database than OLTP transactions.
3. Rows Selected: Although a DSS query tends to summarize information, it may still return more rows than an OLTP query.
4. Throughput: The number of SQL statements executed during the snapshot is expected to be higher in OLTP than DSS.
5. Number of Locks Held: DSS transactions are typically larger and longer than OLTP transactions so we expect more locks are held during the execution of a DSS transaction than an OLTP transaction.
6. Ratio of Using Indexes (%): We expect the ratio of data pages obtained from indexes versus the pages obtained from other database objects, such as tables, in order to satisfy a query to be higher in an OLTP workload than DSS.
7. Number of Sorts: DSS transactions typically perform a larger number of sorts than OLTP transactions.
8. Average Sort Time: Sorts in DSS transactions are usually more complex than the sorts performed in OLTP transactions so they usually take longer time to complete.

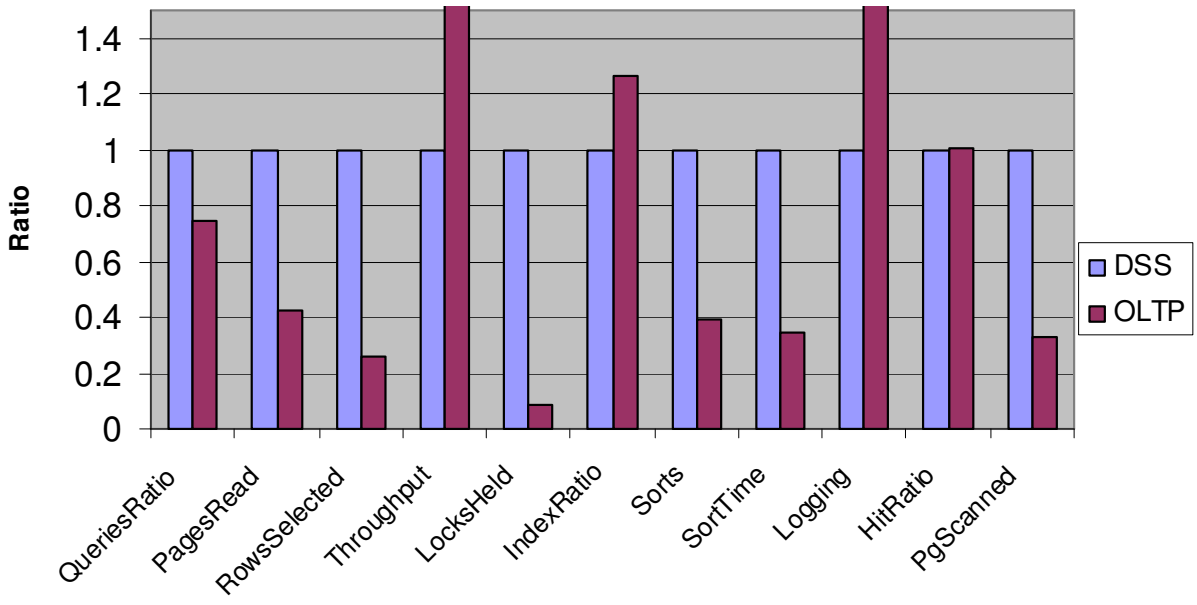


Figure 8. Candidate attributes for snapshot objects

9. Logging: This denotes the number of pages read/written from/to the log file of the database. An OLTP workload generates more logging activity than a DSS workload because of the read/modify nature of the OLTP transactions.
10. Hit Ratio (%): OLTP workloads have a higher degree of locality than DSS workloads and hence OLTP workloads may experience a higher hit ratio on the buffer pool cache area.
11. Pages Scanned: DSS applications typically access large numbers of sequential pages due to the substantial amount of full-table/index scan operations. OLTP applications typically access relatively few random pages.

We consider the *Browsing* and *Ordering* profiles defined in the TPC-W benchmark [91] as examples of DSS and OLTP workloads, respectively. Figure 8 shows the relative values, with the DSS values normalized to 1, for a set of candidate attributes. The values are derived from experiments with the TPC-W workloads on DB2 Universal Database.

System-Dependence	Snapshot Attribute	Weight
Low	Queries Ratio	1.0
	Pages Read	1.0
	Rows Selected	1.0
	Pages Scanned	1.0
	Logging	1.0
Medium	Number of Sorts	0.75
	Ratio of Using Indexes	0.75
High	Sort Time	0.3
	Number of Locks Held	0.3

Table 7. Categorizing the snapshot attributes

The candidate attributes are all easily obtainable by the DB2 Snapshot Monitor and most of them, as illustrated in Figure 8, are relevant. Based on the selection criteria discussed above, we eliminated throughput and hit ratio. Throughput is dependent on the current system utilization and the presently available system resources such as CPUs and memory. Hit ratio is strongly affected by the DBMS configuration, which can include buffer pool sizes and the assignment of database objects to these pools.

The remaining attributes are not equally system-independent. Therefore, as shown in Table 7, we group the attributes into three classes based on their degrees of system-dependence and assign different weights to each class of attribute to reflect their significance to the classification process. We arbitrarily assign weights of 1.0, 0.75, and 0.3 to low-, medium-, and high-dependence attributes, respectively[7]. Queries Ratio, Pages Read, Rows Selected, Pages Scanned, and Logging are the least sensitive to changes in the system configuration. Number of Sorts and Ratio of Using Indexes are somewhat sensitive to configuration changes that are likely to occur infrequently, such as changing the current available set of

[7] These weights are independent of any product or system settings we are using. Any other reasonable numbers that serve in ranking the attribute classes are acceptable.

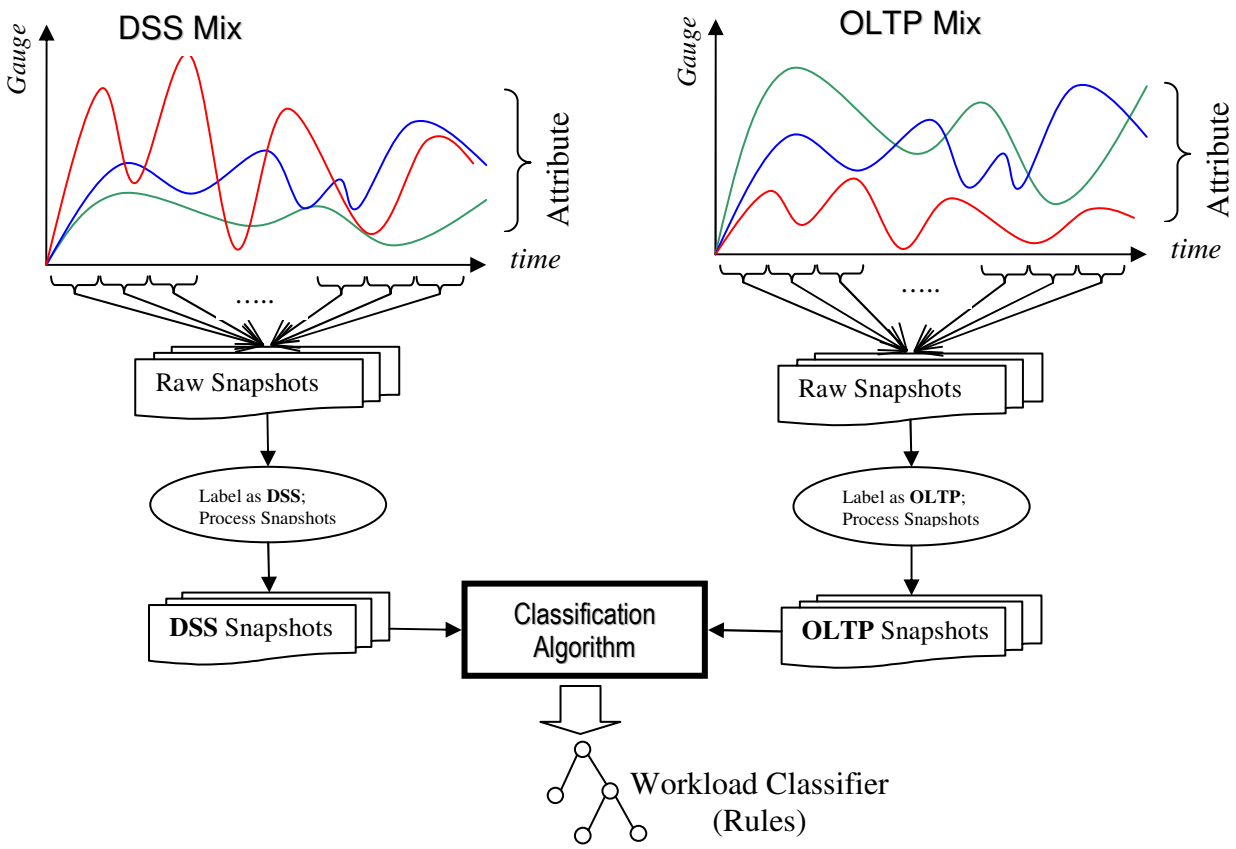


Figure 9. The methodology of constructing a workload

indexes or views in the database schema. Sort Time and Number of Locks Held are the most sensitive to changes in the system configuration and hence they are assigned the lowest weights[8].

[8] Number of Locks Held is dependent on the isolation level, the lock escalation, the application activity, and the application design.

4.3.3 Methodology

Figure 9 illustrates the process of constructing a workload model classifier. We run sample DSS and OLTP workloads and collect sets of snapshots for each one. We label the snapshots as OLTP or DSS and then use them as training sets to build the classifier. We need to choose a snapshot interval such that there are sufficient training objects to build a classifier and the interval is large enough to contain at least one completed SQL statement.

With a snapshot interval of one second, we observed that many SQL statements complete within that size interval in an OLTP workload. This is not the case, however, for DSS workloads that contain complex queries that are too long to complete within one

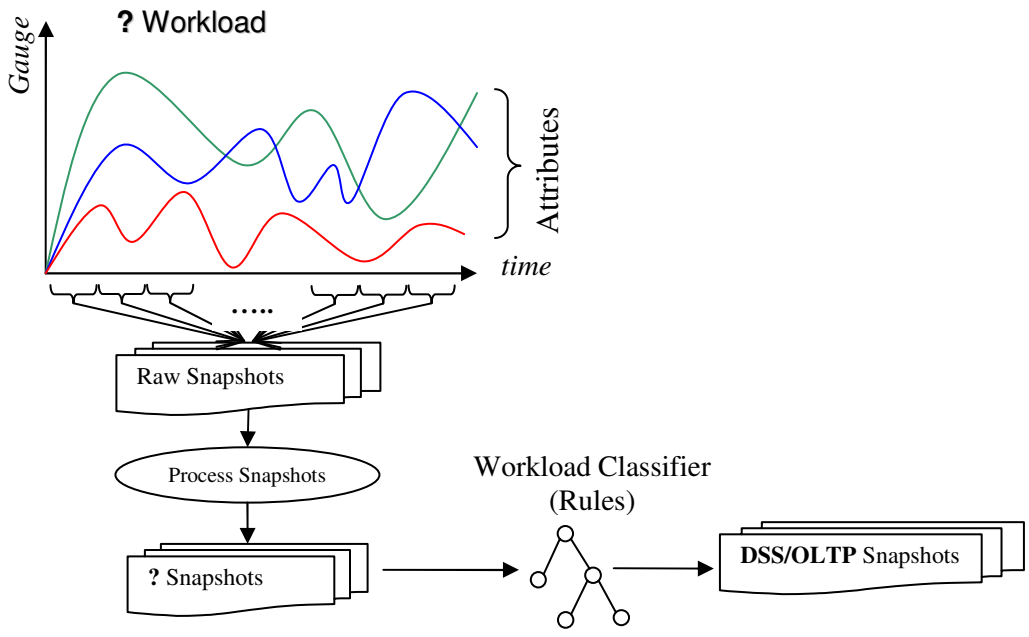


Figure 10. Using the workload classifier to identify unknown workload mixes

second. We therefore dynamically resize the snapshots by coalescing consecutive one-second raw snapshots until we encompass at least one statement completion. We then normalize the consolidated snapshots with respect to the number of SQL statements executed within a snapshot. Consequently, each normalized snapshot describes the characteristics of a single SQL statement. During this training phase, we usually run each workload type (DSS and OLTP) for about 20 minutes, producing a total of 2400 one-second, raw snapshots to process.

After training, we use the generated classifier to identify the OLTP-DSS mix of a given workload as shown in Figure 10. We run the new workload for about 10-15 minutes (producing 600-900 raw snapshots) and produce a set of consolidated snapshots as described above. We then feed these snapshots to the classifier which identifies each one as either DSS or OLTP, supporting this decision by a confidence value between 0.0 and 1.0 that indicates the probability that the class of the snapshot is predicted correctly. For more reliability, only snapshots with high confidence values, greater than 0.9, are considered. On average, we observed that over 90% of the total snapshots examined satisfy this condition. Eventually, we compute the workload type concentration[9] in the mix, C_t, as follows:

$$C_t = \frac{N_t}{S} \times 100$$

where $t \in \{\text{DSS, OLTP}\}$, N_t is the number of snapshots that have been classified as t, and S is the total number of snapshots considered.

[9] In the remainder of the paper, we will sometimes express this concentration exclusively in terms of the DSS percentage (or *DSSness*) in the mix. The OLTP percentage is the complement of the DSSness, that is, 100 – DSSness.

Workload	OS	DB Scale	Memory	CPU	Remarks
TPC-W	Windows® 2000 Professional	10,000 items	512 MB	Pentium III, 733 MHz	100 Emulated Browser; all profiles
TPC-H	Windows® 2000 Server	1GB	512 MB	8-way Pentium II 200 MHz	Throughput test; 25 streams
TPC-C	Windows NT® Server 4.0	100 Warehouses	3 GB	4-way Pentium II 200 MHz	

Table 8. Benchmark settings used with DB2 Universal Database Version 7.2.

Settings used in the DB2 Intelligent Miner	
Maximum Tree Depth	No Limit Imposed
Maximum Purity of Internal Node	100
Minimum Records Per Internal Node	5
Attribute Weights	See Table 7
Error Matrix	None

Table 9. Parameters settings used for the SPRINT classification algorithm

4.4 EXPERIMENTS

Initially, we constructed two classifiers for our experiments. *Classifier (O, B),* was built using the TPC-W Ordering and Browsing profiles as the OLTP and DSS training workloads, respectively. *Classifier (C, H)* was built using the TPC-C and TPC-H benchmarks as the OLTP and DSS training workloads, respectively. We ran each training workload for approximately 20 minutes and collected snapshots every second. The important properties of the experimental setup for these runs are summarized in Table 8 and Table 9.

Figure 11 shows the pruned decision tree for *Classifier(O, B).* The appearance of the Queries Ratio attribute at the root of the tree reveals its importance in the classification process. On the other hand, some attributes, namely, Logging, Number of Sorts, and Sort Time, are no longer part of the decision tree since they have a limited role in distinguishing between

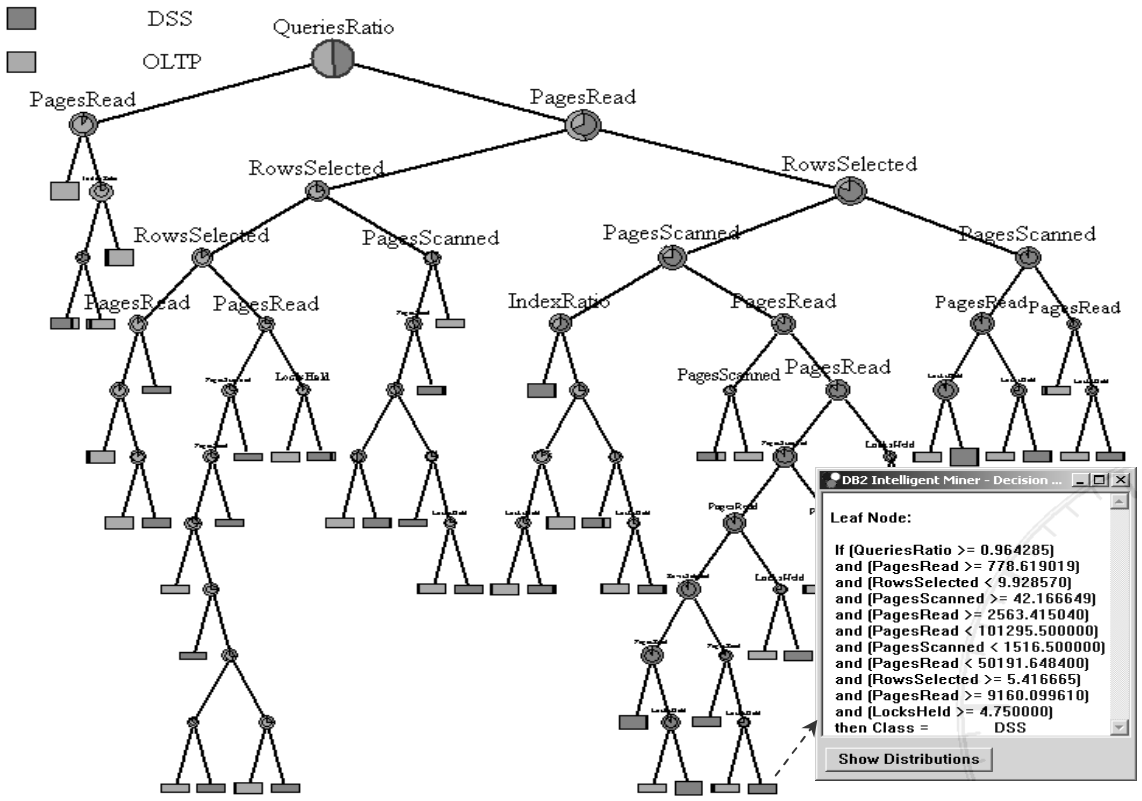


Figure 11. The pruned decision tree for Classifier(O, B). A classification rule is shown.

DSS and OLTP snapshots. The appearance of the Number of Locks Held attribute at lower levels of the tree reflects its low significance in the classification process. This outcome might be partially influenced by the lesser weight we assigned to it.

Figure 12 shows the decision tree of *Classifier(C, H)*. It consists of a single node, namely a test against the Queries Ratio attribute. Apparently, this single test is sufficient to distinguish between TPC-H and TPC-C since the two workloads lie at the extreme ends of the DSS-OLTP spectrum.

We conducted three sets of experiments to evaluate the classifiers. The first set of experiments evaluates the prediction accuracy of the classifiers by inputting new samples from the training workloads. The second set of experiments evaluates the robustness of the classifiers with respect to changes in the mix concentration of the initial workloads,

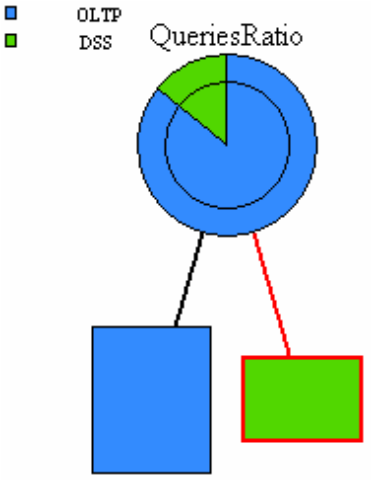


Figure 12. The classification tree of Classifier(C, H)

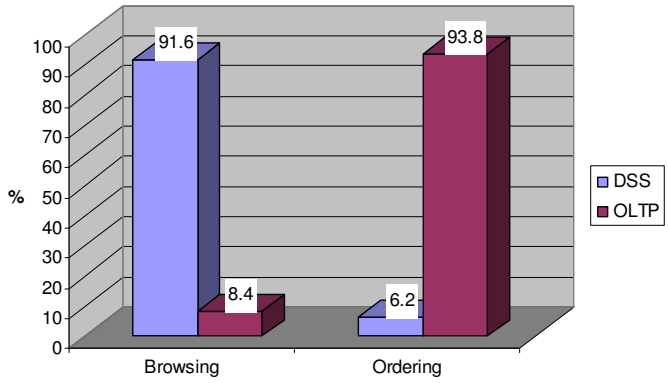


Figure 13. Classifier(O, B) identifying Browsing and Ordering workloads

and with respect to changes to the system configuration. The third set of experiments examines the genericness of the classifiers, that is, their ability to recognize the mix of a different workload. We use both benchmark workloads and industry-supplied workloads in these experiments. The benchmark workloads were run on DB2® Universal Database™ Version 7.2 [49]. The core parameters for the workloads are shown in Table 8.

4.4.1 Prediction Accuracy

Figure 13 shows the results of testing *Classifier (O, B)* against test samples drawn from the Browsing and Ordering profiles. We use the standard three-fold, cross-validation . It shows that *Classifier(O, B)* reports that approximately 91.6% of the snapshots in the Browsing workload are DSS while the rest, 8.4%, are OLTP, whereas it reports that approximately 6.2% of the snapshots in the Ordering workload are DSS while the rest, 93.8%, are OLTP. Similarly, when we applied *Classifier(C, H)* on test samples drawn from TPC-C and TPC-H, it reported that the samples were 100% OLTP and 100% DSS, respectively. Based on our understanding of the characteristics of these standard

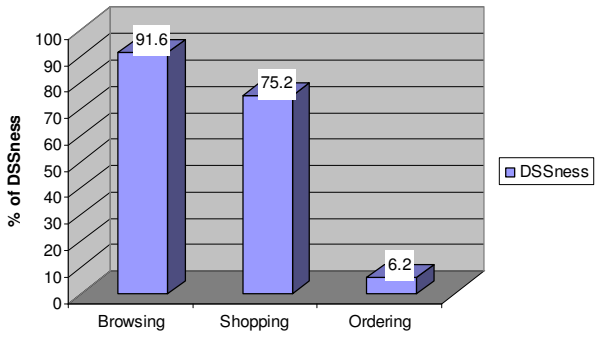


Figure 14. Identifying the Shopping profile.

workloads, and on their description in the benchmark specifications, these results meet our expectations.

4.4.2 Robustness

We use the Shopping profile, a third mix available in TPC-W, to evaluate the ability of the workload classifiers to detect variation in the type intensity of a workload[10]. As seen in Figure 14**,** *Classifier (O, B)* reports 75.2% of the Shopping profile is DSS, which means that the Shopping is closer to Browsing than Ordering. This finding matches the TPC-W specifications, which leads us to believe that the classifier has effectively learnt the characteristics of the TPC-W workload and is able to accurately sense variation in the workload type intensity.

We also examine a classifier's tolerance to changes in the system configurations. For the construction of the classifiers discussed above we ran the training workloads with DB2 Universal Database under the default configuration and with 512MB of main memory. We then tested these classifiers against workloads run on a poorly configured

[10] Note that our classifiers have never been trained on the Shopping profile.

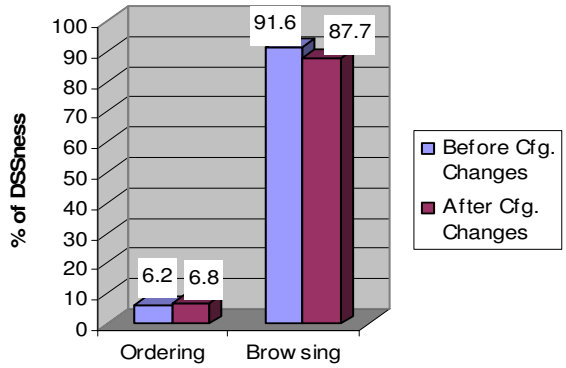


Figure 15. Classifier(O, B) is robust against changes in the system configuration

DB2. Specifically, and in order to cause a dramatic confusion to the classifiers, we ran the Browsing profile on a system configured for OLTP and we ran the Ordering profile on a system configured for DSS. Furthermore, the total memory available for the DBMS was reduced to 256MB in order to cause additional impact on the system. Figure 15 shows that, even under these changes, *Classifier (O, B)* still shows high degrees of accuracy and tolerance to system changes. The predictions reported under the changed system configurations deviate from those of the original by 1%-4%, which is not significant.

4.4.3 Genericness of Classifier(C, H) *and* Classifier(O, B)

In order to evaluate the general usefulness of the classifiers, we test whether a classifier trained on particular workload mixes can be used to recognize the workload mixes of another workload. We tested *Classifier(C, H)* and *Classifier(O, B)* with both benchmark-generated workloads and industrial workloads.

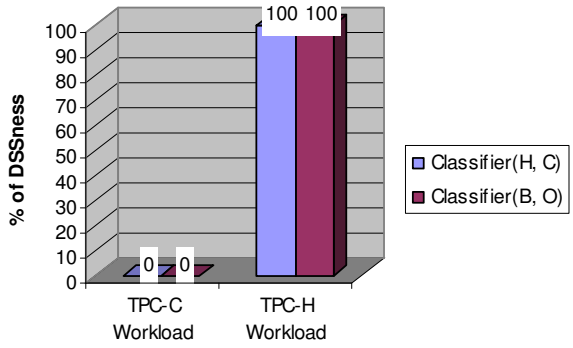


Figure 16. Classifier(C, H) and Classifier(O, B) identifying TPC-C and TPC-H

4.4.3.1 Benchmark Workloads

Figure 16 shows that both *Classifier (O, B)* and *Classifier(C, H)* can accurately identify the workload types in the TPC-C and TPC-H workloads. Figure 17 compares the prediction accuracy of the two classifiers against the three mixes of TPC-W. *Classifier(C, H)* shows poor results due to its simple, single rule derived from the two extreme workloads. We conclude from these results that a single rule is not good enough to distinguish between the mixes of a moderate workload like TPC-W.

4.4.3.2 Industrial Workloads

Our industrial workloads are samples provided by three global investment banking firms, which we identify simply as *Firm-1, Firm-2*, and *Firm-3*[11]. They each provide online financial services including investment research tools and functions for creating and

[11] We appreciate IBM's help in getting production workloads from these major firms. The firms prefer to remain anonymous at this time.

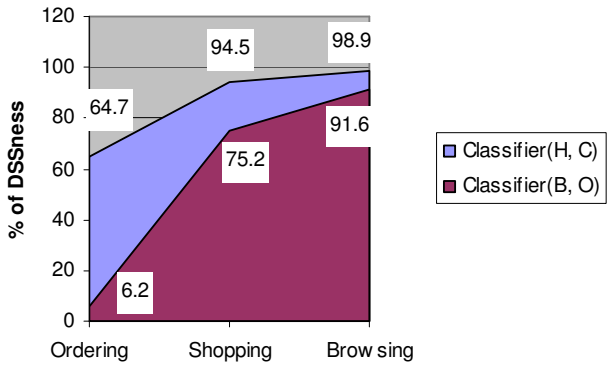


Figure 17. Classifier(C, H) and Classifier(O, B) identifying the profiles of TPC-W

tracking orders. Based on descriptions of the applications provided by the firms, we extrapolate that the characteristics of workloads of these firms resemble the TPC-W profiles workloads. Therefore, we assume that *Classifier(O, B)* is the most appropriate classifier for their workloads. Table 10 summarizes the results of our experiments with the industrial workloads using all of our classifiers (hybrid classifiers will be explained in the next section).

Firm-1 provided several workload samples from an online decision support system that helps investors and shareholders to get the most recent information about the market status in order to help them balance their portfolios and make knowledgeable financial decisions. We noticed a resemblance between the characteristics of *Firm-1*'s workload and of the Browsing profile so we tried identifying this workload type using *Classifier(O, B).* As shown in Table 10, *Classifier(O, B)* reported 90.96% of DSSness, which meets our expectations. On the other hand, *Classifier(C, H)* failed in its identification (62.77% DSSness).

Firm	DBA's Opinion	Classifier(C, H)		Classifier(O, B)		HC		GHC			
		DSS	OLTP	DSS	OLTP	DSS	OLTP	HD	LD	LO	HO
1	Peak DSS	*62.77%*	*37.23%*	90.96%	9.04%	89.79%	10.21%	0%	88%	11%	1%
2	Peak DSS	100%	0%	98.7%	1.3%	100%	0%	0%	98.7%	1.3%	0%
	Peak OLTP	*100%*	*0%*	0%	100%	0%	100%	0%	0%	100%	0%
3	Peak DSS	100%	0%	100%	0%	100%	0%	0%	100%	0%	0%
	Peak OLTP	3.13%	96.87%	0%	100%	0%	100%	0%	0%	90.62%	9.38%

Table 10. Recognition of Industrial Workloads Using All Types of Classifiers

Firm-2 provided samples from both DSS and OLTP workloads. The DSS workload is characterized by complex queries accessing a large fact table (over 3 million rows) and performing join operations with five other small tables. The OLTP workloads, on the other hand, consist mostly of transactions that involve INSERT, UPDATE, and DELETE SQL statements and many simple SELECT statements. Table 10 shows the concentration of DSS work in *Firm-2*'s workloads reported by both classifiers. *Classifier(C, H)* was able to correctly identify the DSS concentration in the DSS workload but failed with the OLTP workload. This failure is again due to the simplicity of this classifier, which relies solely on the ratio of the queries (that is, Queries Ratio attribute) in the mix. The OLTP workload was mistakenly classified as 100% DSS because it contained a substantial number of queries. *Classifier(O, B)*, on the other hand, correctly identified the *Firm-2*'s workload types, which is another indication of the need for a more complex decision tree with more multiple-attribute rules in the general case.

Firm-3 provides customers with a set of DSS-like functions to search for stock information and a set of OLTP-like functions to place orders and manage accounts. They also include administrative tasks such as making money transfers among accounts, and

changing account and trading passwords. The different DSS and OLTP samples collected from this firm were collected in a more controlled environment as we monitored test systems, which made it relatively easy to determine when to collect relatively pure DSS and OLTP workload mixes. Table 10 shows that *Classifier(O, B)* and *Classifier(C, H)* successfully identify the DSS workload (DSSness = 100%) and the OLTP workload (OLTPness > 95%) of *Firm-3*.

4.4.4 Constructing Generic Classifiers

Notwithstanding the success of *Classifier(O, B)*, the above results obtained from assessing the genericness of the two classifiers lead us to believe that a classifier trained on a particular workload should not be expected to be universally good at identifying other workloads, especially if the other workloads have different characteristics. Every workload is a mix of its own set of SQL statements with their own characteristics. Nevertheless, we argue that we can construct a more generic classifier, a *hybrid* classifier, by training it on different flavors of DSS and OLTP mixes in order to derive more generic rules that can recognize a wider range of workloads. We believe that a hybrid classifier can be made more generic by training it on different samples drawn from different flavors of DSS and OLTP workloads. Such training should empower the prediction accuracy because the classifier would attempt to come up with rules that could take into account a wider variety of different workload characteristics. In the subsequent sections we describe two hybrid classifiers we built and evaluated, namely the Hybrid Classifier (*HC*) and the Graduated Hybrid Classifier (*GHC*).

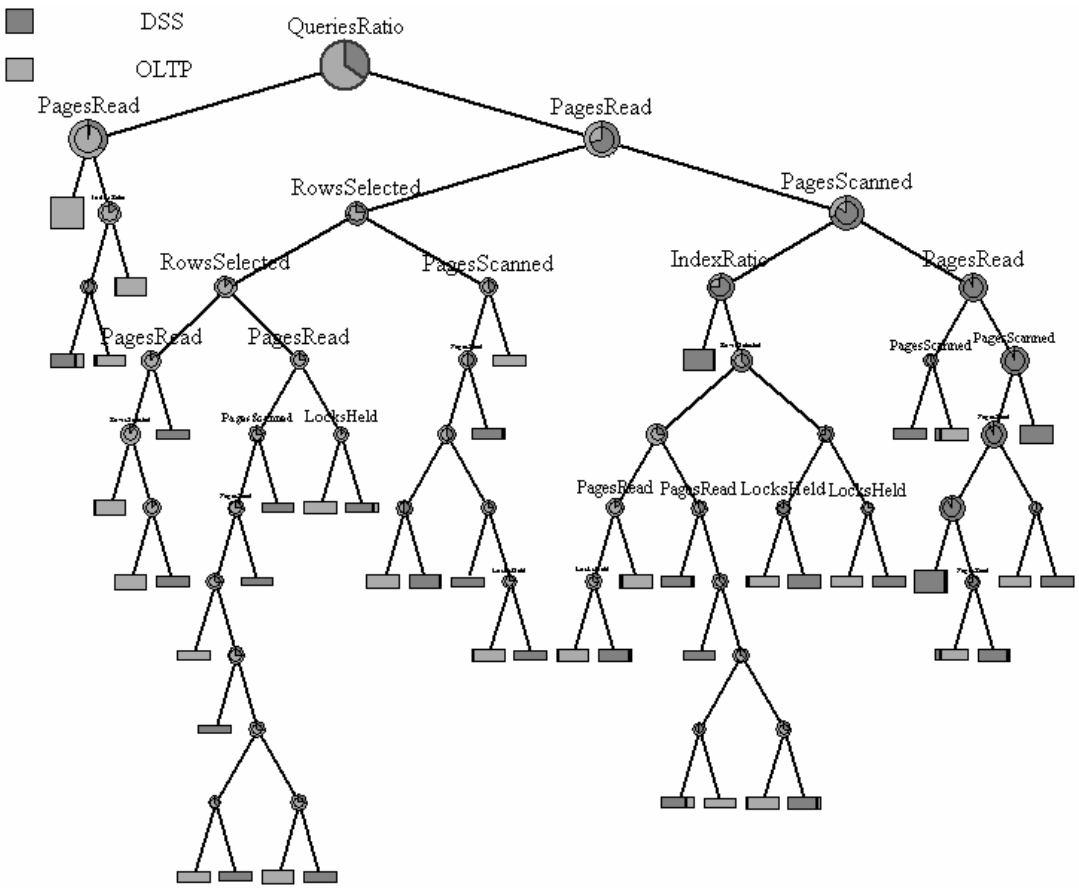


Figure 18. The decision tree of the hybrid classifier (HC)

4.4.4.1 Hybrid Classifier (HC)

The *hybrid classifier (HC)* is trained on different samples drawn from different characteristics, or flavors, of DSS and OLTP workloads. This should improve the prediction accuracy because the classifier would attempt to come up with rules that take into account a wider variety of workload properties. We consider the Browsing and the TPC-H workloads as flavors of DSS and the Ordering and the TPC-C workloads as flavors of OLTP. Figure 18 shows the pruned decision tree of *HC*, which looks structurally similar to the pruned tree of *Classifier(O, B)*, but is different with respect to its rules.

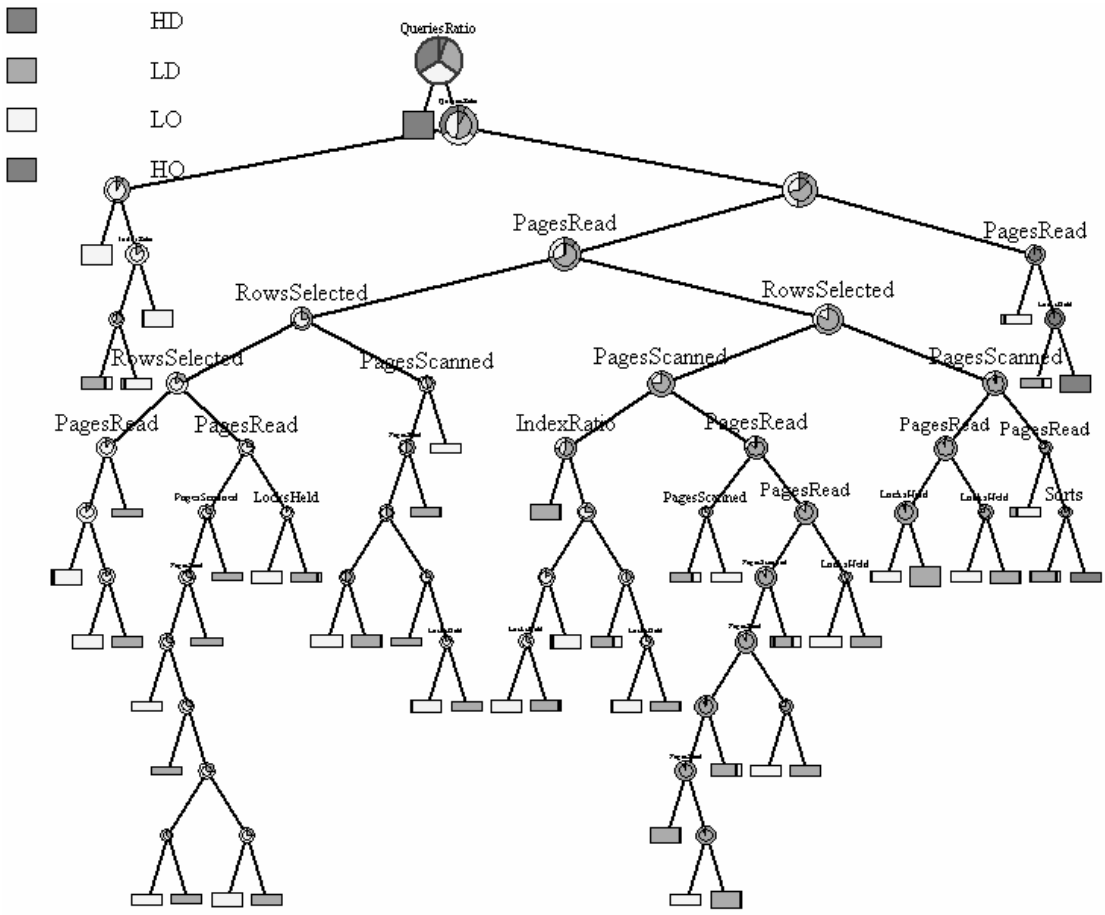


Figure 19. A snapshot of the GHC tree classifying four types of workloads.

4.4.4.2 Graduated Hybrid Classifier (GHC)

For the purpose of effectively configuring a system, it is useful to distinguish between a *heavy DSS* (HD) workload, such as TPC-H, and a *light DSS* (LD) workload, such as the Browsing profile. The same thing is true for a *heavy OLTP* (HO) workload, such as TPC-C, and a *light OLTP* (LO) workload, such as the Ordering profile.

The *Graduated Hybrid Classifier* (GHC) improves upon *HC* by explicitly recognizing a wider variety of workloads, specifically classes HD, LD, LO and HO (Figure 19). *GHC* demonstrates the ability of our methodology to devise a classifier whose rules can

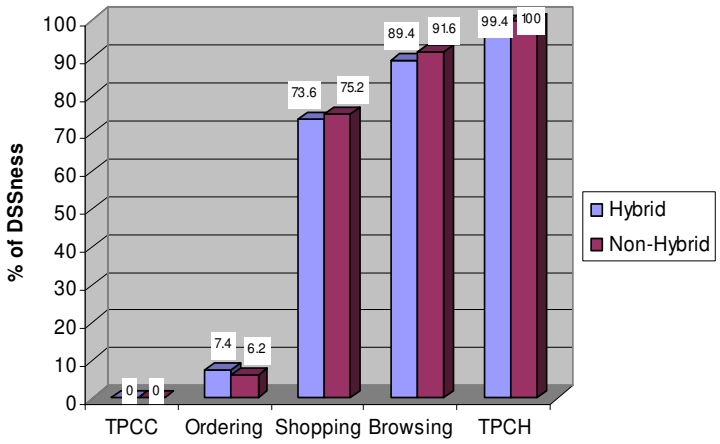


Figure 20. Prediction accuracy of HC

identify finer differences among workloads. In other words, our methodology is able to handle the case of multiple workload types.

We hypothesize that the DSS and OLTP percentages reported by the *HC* are the sums of the HD and LD percentages, and HO and LO percentages reported by the *GHC*, respectively. The results of our experiments validate this hypothesis, as we explain next.

4.4.4.3 Evaluating the Generic Classifiers

We compare the performance of the *HC* and *GHC* with the results reported by the specialized classifier (*Classifier(C, H)* or *Classifier(O, B)*) that we found to be the best at identifying a particular workload sample. We again tested our classifiers with both benchmark-generated workloads and industrial workloads.

4.4.4.3.1 Benchmark workloads

Figure 20 shows the prediction accuracy of *HC,* tested on different testing samples drawn from the various benchmark workloads. The reported DSSness percentage is extremely close to what was reported by each workload's genuine classifier. Figure 21 shows the

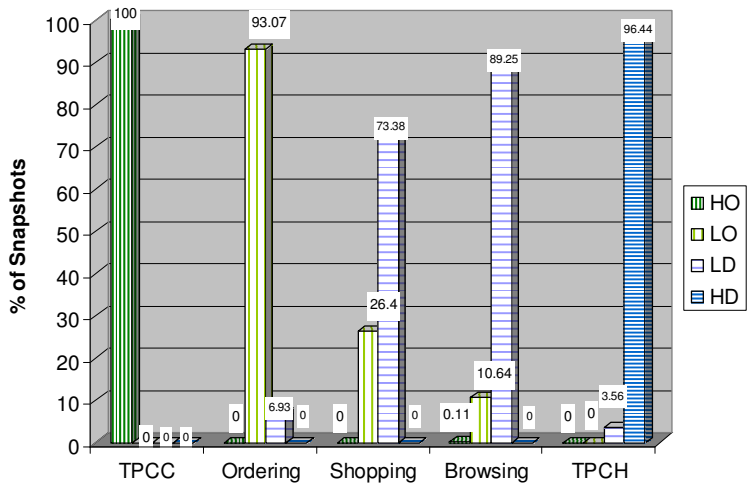


Figure 21. GHC's analysis of TPC-generated workloads

results of the *GHC*'s analysis of the various TPC-generated workloads. This analysis decomposes each workload into four components: HO, LO, LD, and HD. Note how it is rare to observe any HD or HO in the moderate TPC-W profiles. Similarly, the presence of the light workloads of TPC-W profiles is very little in the extreme workloads of TPC-C and TPC-H (there is 3.56% of LD in the TPC-H, which is very small). We conjecture that the more varieties of workload types with which the hybrid classifiers are trained, the more generic and useful they become.

4.4.4.3.2 Industrial Workloads

The results of our experiments in Section 4.4.3 confirmed our assumption that *Classifier(O, B)* is the most appropriate classifier for identifying the workloads of the three e-business firms. Therefore, we should compare the performance of the two generic classifiers with the performance of *Classifier(O, B)*.

As seen in Table 10, *Classifier(O, B)* reported 90.96% of DSSness in Firm-1's peak DSS workload, and *HC* reported a similar percentage, 89.79%. *GHC* also reported a similar

percentage of DSSness, namely 88%, but made the further distinction that this was all light DSS (LD). *GHC* also indicated that the OLTP portion in *Firm-1*'s workload is actually a mix of LO (11%) and HO (1%).

Table 10 shows that all classifiers, including *HC,* assigned a high DSSness (almost 100%) to *Firm-2*'s peak DSS workload. However, *GHC* makes the further differentiation that the workload is all LD, which is correct. Likewise, *Classifier(O, B), HC* and *GHC* all recognized the high percentage of OLTPness (100%) in *Firm-2*'s peak OLTP workload. With respect to Firm-3's workloads, we found that all of the four classifiers were able to correctly recognize *Firm-3*'s peak DSS and peak OLTP workloads (Table 10). *GHC* makes the further distinction that the OLTP workload is composed of 90.62% of LO and 9.38% of HO.

GHC is more practical because it gives a qualitative dimension to what is being reported as DSS and OLTP. We also observe that the total sum of HD and LD workloads reported by the *GHC* is almost equal to the DSSness reported by the *HC*. Similarly, the total sum of HO and LO workloads, reported by the *GHC*, is almost equal to the OLTPness reported by the *HC*. The results lead us to conclude that *GHC* produced good, finer classification rules that are able to distinguish among the various shades of DSS and OLTP workloads.

4.5 SUMMARY

In order to manage their own performance automatically, autonomic DBMSs must be able to recognize important characteristics of their workload, such as its type. In this chapter, we present a methodology by which a DBMS can learn how to distinguish between today's two dominant workload types, namely DSS and OLTP. Our

methodology uses classification techniques from data mining to analyze performance data available from a DBMS to build a classifier for that workload. Once built, the classifier can be used to detect if the workload shifts from one type to another and to evaluate the relative intensity of each type at a point in time.

We demonstrate our methodology by creating and evaluating two classifiers. One classifier, *Classifier (O, B),* is built using the TPC-W Ordering and Browsing profiles as the OLTP and DSS training sets, respectively. The second classifier, *Classifier (C, H)*, is built using the TPC-C and TPC-H benchmark workloads as the OLTP and DSS training sets, respectively. The key difference between the two classifiers is the complexity of their decision trees. *Classifier (C, H)* consists of one single-attribute rule, namely a test against the Queries Ratio, while *Classifier (O, B)* uses several multi-attribute rules to distinguish DSS from OLTP. We found the single-attribute classifier did not identify general workloads as well as the multi-attribute classifier.

We present three sets of experiments with the classifiers. The first set of experiments shows the validity of the classifiers since they are able to accurately recognize different test samples from their base workloads. The second set of experiments shows the robustness of the classifiers. *Classifier (O, B)* is able to accurately determine the relative concentration of DSS and OLTP work within the Shopping profile, which is a variation of its base workloads. *Classifier (O, B)* is also shown to be able to accurately recognize its base workloads under different system configurations. The third set of experiments examines the genericness of the classifiers. In these experiments we used both benchmark and industrial workloads. We found that *Classifier (C, H)*, because of its trivial decision tree, was not able to adequately recognize some general workloads. *Classifier (O, B)*, on the other hand, had good results with both the benchmark and industrial workloads.

We believe that our experiments indicate that, despite the fact that every workload is a mix of its own set of SQL statements with their own characteristics, we can construct a generic classifier that is able to recognize a wide range of workloads. Therefore, we presented and evaluated two generic workload classifiers for automatically recognizing the type of the workload.

The *Hybrid Classifier (HC)* was constructed with training sets that represent a wider range of different characteristics, or flavors, of DSS and OLTP workloads. Our experiments show that such a training method improves the performance of the *HC* over our previous classifiers because it forces the creation of more sophisticated rules that are capable of recognizing the different flavors of DSS and OLTP work.

The *Graduated-Hybrid Classifier (GHC)* improves upon *HC* by also reporting on the workload flavors (*light* and *heavy*), and their concentrations, that constitute these DSS and OLTP portions in the analyzed sample. In addition to the practical benefits of being able to make finer distinctions, *GHC* demonstrates that our method is able to construct classifiers for more than two workload types.

Our experiments with benchmark workloads and the industry-supplied workloads confirmed that the total DSSness reported by the *HC* is almost equal to the summation of its components, the HD and LD, reported by the *GHC*. Similar results were observed with respect to the OLTPness and its components, HO and LO. This reflects the accuracy of the predictions of the hybrid classifiers.

The good results obtained from testing the generic classifiers make us believe that it is feasible to consider incorporating them into a DBMS to tune, or at least help tune, the system. DB2 Universal Database v8.1, for example, includes a Configuration Advisor that defines settings for critical configuration parameters for a DB2 database based on

Figure 22. The type of the workload is yet a decision that human has to make

workload characterization, and system environment. While the Configuration Advisor is able to automatically detect its physical system environment through programmatic means, it requires descriptive input from either a human operator or a calling application to define characteristics of the workload environment (Figure 22). A workload classification engine, such as the one described in this chapter, would automate the classification process, obviating the need for the user to provide some of the key input to the Configuration Advisor.

Automatic classification within the Configuration Advisor would allow for the automatic generation of settings for operational parameters such memory allocations (sort, buffer pools, lock space, communication buffers, etc), parallelism degrees, aggressiveness of page cleaning or prefetching, and query optimization depth, whose internal modeling are a function of the workload classification. In many cases, it is

reasonable to expect the classifier to more accurately identify the operational workload than a human operator.

Note that our approach is independent of any specific DBMS or classification tool. Moreover, and based on the criteria we set, the snapshot attributes we selected are the result of a comprehensive study of more than 220 performance variables. These variables are commonly available in today's commercial DBMSs such as DB2 and Oracle [70] in order to allow DBAs observe and diagnose the performance of the system.

Workload classifiers can be useful and put into practice in different ways. One approach of incorporating the classifiers into DBMSs is to provide a set of prefabricated, ready-to-use workload classifiers for different popular workload types. A second approach is to adopt one of the hybrid classifiers that is trained on a wide variety of workloads.

Furthermore, a feedback mechanism can be 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. 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.

The classification methodology we used follows the guidelines of the characterization framework presented in the previous chapter. For example, in the *Requirements Analysis Phase*, we decided to do resource-oriented characterization for the low-level performance attributes (e.g., # of pages read and avg. sort time) of SQL statements (*Basic Workload Components*). In this chapter, we described what workload attributes we need to analyze and the criterion behind their selection. In the *Model Construction Phase*, we used the

DBMS monitors to collect performance data. These data were preprocessed and stored in traditional text files as their volume does not warrant the use of a DBMS. In our analysis, we ultimately used data mining with some basic statistical analysis. The final workload model is represented by a decision tree produced from the classification algorithm. In the *Validation Phase*, we tested this model using independent workload samples in order to validate the prediction accuracy of the classification tree.

CHAPTER 5 WORKLOAD PREDICTION

5.1 PROBLEM AND MOTIVATION

In the previous chapters, we explained the need for autonomic systems that manage themselves in light of the characteristics of their workload. We specifically discussed how important it is for a complex system such as a DBMS to automatically recognize the type of its workload, namely whether it is OLTP or DSS, in order to tune its performance. We also presented a classification methodology by which the DBMS can identify the type of the workload automatically.

However, identifying the type of the workload is just the beginning. A DBMS may experience changes in the type of workload it handles during its normal processing cycle. For example:

- when new data are rolled in to or rolled out from a warehouse vs. when analysts are querying it.
- a bank may experience an OLTP-like workload by executing the traditional daily transactions for most of the month, while in the last few days of the month, the workload becomes more DSS-like due to the tendency to issue financial reports and run long executive queries to produce summaries.
- In the money market, traders may exhibit some daily pattern as they access the information systems of their brokers [92]. For example, in the early hours of the market, traders tend to intensively query the system in order to analyze historical performance of the market and to analyze some candidate stocks. After this phase of analysis, traders may place financial transactions (buying, selling, etc.) for the rest of

the morning session. At noon, and during lunch hour, they keep querying the system in order to keep track of the progress of their portfolio. The early afternoon session may be a mix session of placing new transactions and monitoring the performance of the current holdings. In the late hours of the day, near the closing time of the market, traders tend to aggressively submit orders in order to close their financial positions or to open new ones for the next day. After the market closes, the system dominantly experiences a DSS-like workload as traders analyze the day's performance and assess the status of their portfolio.

We believe that such changes can be predictable by analyzing historical data. Therefore, it is not enough for autonomic DBMSs to identify the current type of the workload, but also to predict when a change in the workload type will occur. We could simply keep the workload classifier activated and monitor the system constantly to sense significant shifts of workloads. However, this approach imposes undesirable overhead and perturbation on the system. We found in our experiments that running the workload classifier reduces the throughput of the DBMS by 10% on average. Moreover, if each newly introduced autonomic feature does not take care to reduce its operational cost, this incremental introduction of features and functions could lead to accumulative overhead that threatens to undermine the very benefits autonomic computing aims to provide.

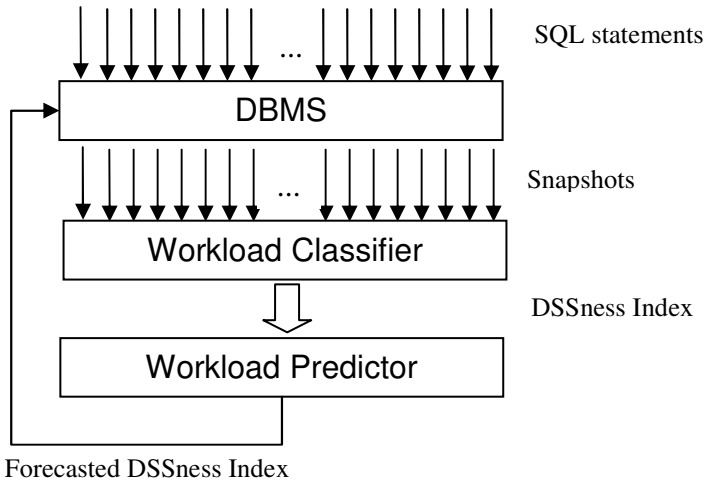


Figure 23. The integration between the workload classifier and predictor

The goal of this chapter 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. It is important to realize that the workload prediction problem builds on top of the work on workload identification, and complements it as illustrated in Figure 23. The workload classifier's job is to assess the DSSness of the workload at a given time. The workload prediction architecture, 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. These thresholds divide the DSSness range into three zones that lead to the identification of three main workload types: OLTP, MIX, and DSS.

This chapter is structured as follows. Section 5.2 discusses the possible prediction approaches that can be used to identify workload shifts. Section 5.3 presents a high-level

description of our approach and our prediction architecture. Sections 5.4, 5.5, and 5.6 provide a detailed description of the core components of the architecture. Section 5.7 describes the various operation modes that a DBMS can adopt in order to be workload-aware. Section 5.8 evaluates the performance of our prediction approach and compares it with other operation modes.

5.2 PREDICTION APPROACHES

Initially, we can think of solving this prediction problem by one of two approaches:

1. ***On-line Prediction***. Some prediction problems that attempt to detect the idle periods in computer systems use on-line prediction techniques [35] that require continuous monitoring of the system as long as it is on-line and operational. This approach aims to forecast (using, for example, moving averages or exponential smoothing techniques) very near-future events such as when a disk becomes idle so the system can spin it down in order to save energy.

   On-line prediction is usually done by constantly collecting data about a single performance index in the system, such as the number of I/O accesses, and by analyzing its fluctuation over time. Such data represent a single time series. Our workload type prediction problem resembles idleness detection problems in some aspects, and differs from them in others. It is similar to idleness detection problems in the sense of using the DSSness index as the single time series that fluctuates over time as a reflection of the change in the concentration of the DSS-OLTP mix in the workload. If the DSSness rises, the higher the intensity of the DSS mix is in the workload.

However, the major difference between our problem and the traditional idleness prediction problem is that the latter counts on monitoring one *basic*, *primitive* performance index collected at run-time in such a way that does not impose significant overhead on the system. In contrast, our DSSness index, produced by the workload classifier, is a metric resulting from a non-trivial analysis of several performance variables (snapshot attributes) collected on-line from an intricate system such as the DBMS. This inevitably causes extra overhead on the system and impairs its performance.

2. ***Off-line Prediction***. Another way of predicting workload type changes is by performing a one-time, off-line analysis. This approach is useful for data that are relatively easy to forecast as they likely exhibit a certain cyclical patterns over a time window (e.g., daily or weekly). Based on our experience, the workload type prediction problem is a good candidate for this approach due to the low volatility of change of workload type in real systems. A change typically occurs over several hours as a result of users' tendencies to run particular types of applications at certain times.

   This one-time, off-line analysis, however, is less trustworthy than the on-line prediction because exceptional behavior may occur during the course of the day in a way that contradicts the suggestion made by the off-line predictor. Consequently, if the DBMS puts absolute trust in the off-line prediction and resets its parameters accordingly then performance could dramatically degrade and the penalty of such a wrong prediction becomes very costly.

5.3 THE PSYCHIC-SKEPTIC ARCHITECTURE

There are three possible *operation modes*[12] under which the DBMS can operate with respect to workload type. The first operation mode is the Default Mode in which the DBMS uses the default, out-of-the-box settings that suit mixed workloads in general. The second mode is *Dominant Workload Mode* in which the DBMS is tuned to suit the dominant workload throughout the day. The third mode is the *Continuous Monitoring Mode* in which the DBMS counts on on-line prediction operations in order to forecast near future shifts in the workload type.

We propose a fourth mode that uses the *Psychic-Skeptic* architecture. This architecture

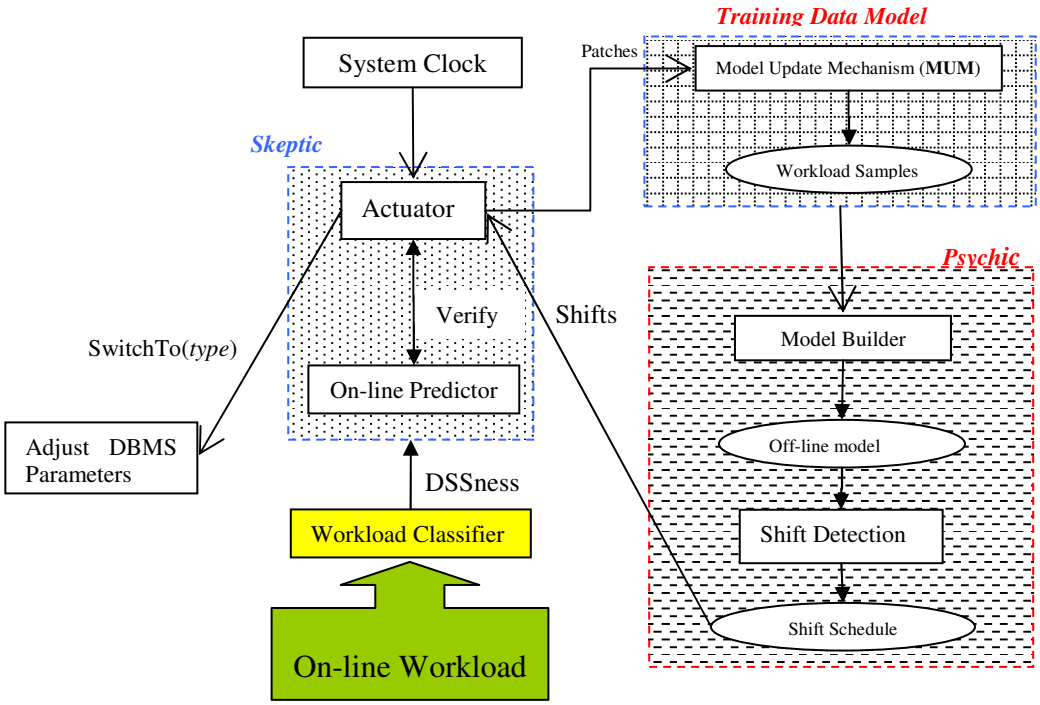


Figure 24. Psychic-Skeptic Architecture

[12] More details about these modes are provided in Section 5.7.

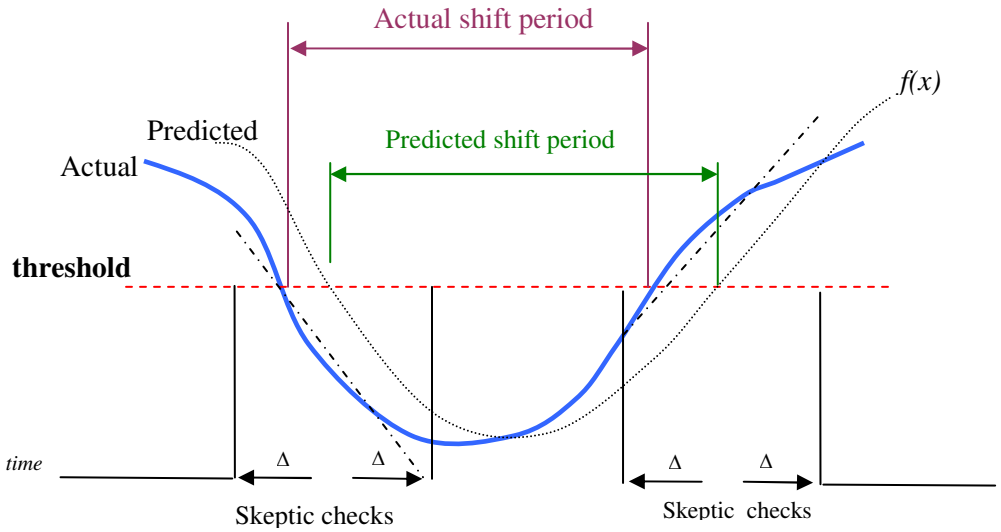


Figure 25. The Skeptic verifies the Psychic's predictions

takes advantage of the combination of the online and off-line predictive approaches in order to make effective, low cost predictions. Note that our focus is on repeatable, daily patterns and we are not concerned with handling bursts that may suddenly occur during the day for some unexpected reason. For example, as a response to an unscheduled, managerial request, the DBA may run a reporting-intensive application in order produce the required summaries. Such bursts are considered exceptions because they do not form a regular pattern in that business.

As depicted in Figure 24, the Psychic-Skeptic architecture consists of three main components: the *TrainingDataModel*, the *Psychic*, and the *Skeptic*. The premise of the prediction architecture is as follows. The Psychic analyzes a daily time series of DSSness 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 DSSness thresholds (see Figure 25). These shifts are passed to the Skeptic who does not give

absolute trust to the Psychic's predictions. Rather, the Skeptic keeps monitoring the system time in order to intercept the nearest upcoming forecasted shift. When the shift is due, the Skeptic validates the shift by performing an on-line, short-term prediction using linear regression. So, if the shift is due at time t, the Skeptic monitors the system for the Δ minutes before and after t, that is the interval $[t - \Delta, t + \Delta]$. The Skeptic does not instruct the DBMS to reset its parameters unless it confirms the trend of the shift using the linear model. In order to keep all prediction models updated, the Skeptic is also responsible for performing regular sampling for the DSSness throughout the day and sends these DSSness patching samples to the TrainingDataModel in order to update the stored training data. The Psychic refreshes its off-line prediction model and its forecasted shift schedule accordingly. This regular update guarantees the adaptability of the architecture 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 is time window over which our prediction architecture 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. Next we describe the global parameters of the prediction architecture.

Module	Parameter	Description	Range	Default Value
Global	MUM	Boolean parameter to activate or deactivate the Model Update Mechanism.	ON or OFF	ON
	monCost	Workload classifier overhead	(0-100)%	10%
	performanceMatrix	DBMS's performance	0.0-1.0	See Table 12
	dss_threshold	DSSness value that lies between the DSS and MIX workload zones	(0-100)%	70%
	oltp_threshold	DSSness value that lies between the MIX and OLTP workload zones	(0-100)%	30%
	min_check_time	# minutes that Skeptic needs to validate a shift	30-1440	30 min.
TrainingDataModel	numScenarios	# of daily scenarios used for training	1-5	3
	numDaysToCompleteUpdate	# of days needed to update a whole scenario	fastUpdateDays – slowUpdateDays	7

Table 11. Parameters of the Psychic-Skeptic Architecture

5.3.1 Parameters of the Architecture

Our prediction system's configuration depends on a number of key parameters. Having these parameters adds flexibility to the architecture and makes it generic and adaptive to the setup of the IT environment. Table 11 summarizes these parameters under two categories: global and TrainingDataModel. Most of these parameters are automatically estimated by the architecture or derived from the computing setup surrounding the DBMS.

Now we provide a brief description of the global parameters that are used across several modules in the architecture. These parameters need to be set once and remain constant afterwards. Other types of parameters will be explained in the context of subsequent sections.

- **Model Update Mechanism (MUM)** is a switch parameter that can be set to ON or OFF in order to respectively enable or disable the Model Update Mechanism feature of the architecture. In general, if the daily pattern is trusted to remain stable, the MUM

	OLTP Settings	MIX Settings	DSS Settings
OLTP workload	1.0	0.5	0.3
MIX workload	0.5	1.0	0.5
DSS workload	0.3	0.5	1.0

Table 12. Performance Matrix

parameter can be OFF. Setting MUM to ON entails a small overhead but guarantees to keep the prediction models up to date and provides immunity against changes in the workload pattern.

- **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.
- **oltp_threshold** and **dss_threshold**. Most of the Current DBMSs are tuned based on identifying three main types of workloads (Figure 22): DSS, OLTP, or MIX. We use the DSSness percentage and the **oltp_threshold** and **dss_threshold** to identify workload shifts as follows:

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

The values of *oltp_threshold* and *dss_threshold* represent the empirical values of the DSSness at which it is worth resetting the DBMS configuration to suit the new workload type. We have empirically found that 30 and 70 are good estimates for the *oltp_threshold* and *dss_threshold*, respectively.

- **performanceMatrix** is a 3x3 matrix. If x and $y \in$ {OLTP, MIX, DSS}, then each entry ($Workload_x$, $Settings_y$) in the *performanceMatrix* is a performance factor that denotes the relative performance of a DBMS, processing workload type x when its settings are suitable for workload type y, to the optimal performance of this DBMS

Day Samples	Weight
Day_0	*1*
Day_1	*2*
Day_2	*4*
⋮	⋮
$Day_{numScenarios-1}$	$2^{(numScenarios-1)}$

Table 13. Training scenarios stored in the TrainingDataModel

when it processes workload *y* under settings suitable for type *y*. Therefore the optimal performance is deemed 1.0 and each entry is a ratio between 0 and 1.0. All cost-benefit analyses use this performance matrix. These performance factors are empirically determined just once by running different combinations of different workloads vs. different DBMS settings. Table 12 shows the empirically estimated performance factors that we use in our experiments.

- **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 size to start with. The final size is eventually determined by the Psychic after analyzing the training data. *min_check_time* is also the minimum size of the time slots used by the Model Update Mechanism (MUM) to patch the historical data (as explained in Section 5.5.3).

Next, we describe the Psychic-Skeptic architecture components in detail. We explain their functions, their parameters, and how they collaborate with each other.

5.4 THE TRAINING DATA MODEL

The TrainingDataModel is a queue-like data structure component (Table 13) that is responsible for storing and managing historical samples, we call them *Scenarios*, of a number of days (*numScenarios*) to train the Psychic and the Skeptic. It is equipped with all functions to add, remove, and edit these data. TrainingDataModel stores *numScenarios* chronologically ordered *scenarios* (full day samples). $Day_{numScenarios-1}$ is the most recent while Day_0 is the oldest. Each day is assigned a weight that is double that of the previous day. Therefore, Day_0 is assigned a weight of 1, Day_1 is assigned 2, Day_2 is assigned 4, etc. $Day_{numScenarios-1}$ is assigned $2^{(numScenarios-1)}$. In order to adhere to the notion of probabilities, we normalize these weights by transforming them to the 0-1.0 scale, as explained next. Assigning weights to the sample days is vital to the quality of the prediction models built in this architecture as such weights put more stress on the most recent observed days than the older ones. 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.

Next, we describe the main functions that the TrainingDataModel provides to the other architecture components.

5.4.1 Predictability Assessment

The Psychic-Skeptic architecture is based on the premise of predictable pattern. In order to devise a mechanism that allows us to verify the existence of a predictable, cyclic DSSness pattern over the *numScenarios* days stored in the training model, we need to view the DSSness samples of *all* days as a single time series. We can then use the

autocorrelation coefficient, r_k, to test the predictability of the DSSness using the following formula [60]:

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

where

k denotes the lag (k=1,2,3,…), which is the length of the daily prediction period (k = *max_time_scale*).

N is the total number of samples collected over *numScenarios* days, N= *numScenarios** *max_time_scale*.

D_i is the DSSness sample number i, where i=1..N.

$\overline{D}$ denotes the mean of the DSSness samples, that is, $\overline{D} = \frac{\sum_{i=1}^{N} D_i}{N}$

r_k is the autocorrelation coefficient whose range 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. In general, we deem r_k >0.5 a strong indication of having a predictable trend. In our experiments, r_k is 0.65 on average.

5.4.2 Model Consolidation

In order to analyze the daily scenarios, we transform them into a compact form that represents all days while taking into account the weight of each day. As shown in Figure

```
Task consolidatingScenarios
   total = 2^numScenarios -1

   consolidatedScenario = new Scenario();

   For t=0 to max_time_scale
      avgDSS = 0;
     For d=0 to (numScenarios-1)
        sample = scenario[d].Sample[t];
        avgDSS += sample.dssness *(d/total);
     Endfor  //for each daily scenario in the TrainingDataModel

     consolidatedScenario.addSample(t, avgDSS);
    Endfor // for each time tick within each scenario

    return consolidatedScenario;
End Task
```

Figure 26. Model Consolidation

26, this consolidated scenario is constructed by calculating $avgDSSness_t$, which represents the weighted average of all $DSSness_{(d,t)}$ samples collected at time t, in scenario number d:

$$avgDSSness_t = \sum_{d=0}^{numScenarios-1} DSSness_{(d,t)} \times w_d$$

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

As we discuss in subsequent sections, the consolidated scenario is needed by numerous components of the architecture. For example, the Psychic builds an off-line prediction model by applying polynomial regression to the consolidated scenario. It also estimates the interval [*earliestCheckTime*, *latestCheckTime*] during which the Skeptic works in order to validate the forecasted shifts. In addition, the dominant workload type is determined by analyzing the consolidated scenario. The Model Update Mechanism (MUM) also uses the consolidated scenario to back-test the performance of the DBMS under different operation modes in order to optimize the MUM parameters.

```
Task ModelPatching (patchingScenario)
   // Patching all scenarios in TrainingDataModel except the most recent one
   // by back propagation
   For d=0 to numScenarios-2
      s1 = scenarios[d];
      s2 = scenarios[d+1];

     // Get each sample in patchingScenario and patch at its time tick
     For i=1 to patchingScenario.numSamples
        s = patchingScenario.sample[i];
        t = s.time;
        s1.sample[t] = s2.sample[t];
      Endfor // for each sample the patchingScenario
    Endfor // for each scenario in TrainingDataModel

    //Now, patch the most recent scenario with samples from patchingScenario
    lastScenario = scenarios[numScenarios-1];

    For i=1 to patchingScenario.numSamples
     s = patchingScenario.sample[i];
     lastScenario.sample[s.time] = s;
    Endfor

End Task
```

Figure 27. Model Patching

5.4.3 Model Update Mechanism (MUM): Patching

The prediction architecture manifests its adaptability to pattern changes by having the Model Update Mechanism (MUM) that can *patch* 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.

Patching occurs by propagating the DSSness samples of a particular scenario to the next older scenario. A $\text{DSSness}_{(d,\ t)}$ sample of day d at time t replaces $\text{DSSness}_{(d\text{-}1,\ t)}$ for d=1..*numScenarios*-2. $\text{DSSness}_{(numScenarios\text{-}1,\ t)}$, which is the most recent scenario in the TrainingDataModel, is patched by the newly DSSness samples collected by the Skeptic. Figure 27 sketches the patching task.

5.4.4 Determining the Dominant Workload

The DBA can run the DBMS with fixed settings that suit the dominant workload type experienced in a business. This type can be systematically determined by analyzing the consolidated scenario derived from the historical data. The TrainingDataModel determines the dominant workload type by scanning the consolidated scenario and constructing a distribution of the DSSness sample types (DSS, MIX, or OLTP). The type that has the highest frequency is deemed to be the dominant one.

Selecting the value of *numScenarios* is a tradeoff between the quality of the off-line model and the pace by which *all* scenarios in the data set can be fully refreshed. Having multiple days in the training set may lead to a more robust off-line prediction model. However, the MUM will take a longer time to patch the entire stack of scenarios according to the propagation algorithm described earlier. Experimentally, we found that *numScenarios*=3 is a reasonable size that produces good prediction quality and high adaptability.

5.5 THE PSYCHIC

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

1. *Off-line Model Generation*. The psychic analyzes historical data stored in the TrainingDataModel and produces the best, least complex polynomial that fits them.

2. *Finding Shifts*. The produced polynomial 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 start and end time of this timeframe, for each shift, is estimated at this stage.
4. *Filtering Shifts*. Not all shifts are good. The Psychic performs a cost-benefit analysis to determine if a shift is worth consideration or if ignoring it would be more beneficial to the overall system performance.
5. *Setting the MUM Parameters*. The MUM assures the validity of prediction models used in this architecture. It makes the prediction architecture less vulnerable to possible changes in the workload pattern over time. In order to achieve this goal efficiently, the MUM needs to optimize its internal parameters in light of the characteristics of the workload.

More details about these tasks are explained in the following subsections.

5.5.1 Off-line Model Generation

The Psychic uses a polynomial as an off-line model. A representative polynomial is generated by applying the polynomial regression algorithm to the consolidated scenario obtained from the *TraniningDataModel*.. In our experiments, the produced polynomials are mostly from the 3rd and 4th degrees. 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 [60]. 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 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 extrapolation using polynomial regression, on the other hand, lends itself to the ease of geometric manipulation (i.e., it is easy to find where the polynomial intersects with certain threshold) and it is an intuitive, compact representation for the workload trend.

```
Task findShifts
  shiftSchedule = new ShiftSchedule();


  // ************ find intersections with dss_threshold *************
  Roots[] = findRoots(offlinePredictionModel - dss_threshold);
  firstDerivative = offlinePredictionModel.derivative();

  For i=1 to roots.numRoots
    slope = firstDerivative.value( round(roots[i]) );

   // Abandon minima and maxima because they are not real shifts.
   If (abs(slope)>0.001 AND roots[i] in [0..max_time_scale])
      shift = new Shift();
     shift.setTime(roots[i]);

    If (slope >0)
           shift.setType(MIX_UP_TO_DSS);
        Else
           shift.setType(DSS_DOWN_TO_MIX);
    Endif

    shiftSchedule.addShift(shift);
   Endif

  Endfor // for each root

  // ************ find intersections with oltp_threshold ***********
  Roots[] = findRoots(offlinePredictionModel - oltp_threshold);

  For i=0 to roots.numRoots
   slope = firstDerivative.value( round(roots[i]) );

   // Abandon minima and maxima because they are not real shifts.
   If (abs(slope)>0.001 AND roots[i] in [0..max_time_scale])
    shift = new Shift();
    shift.setTime( roots[i] );

    If (slope>0)
           shift.setType(OLTP_UP_TO_MIX);
    Else
           shift.setType(MIX_DOWN_TO_OLTP);
    Endif

    shiftSchedule.addShift(shift);
   Endif

  Endfor // for each root

  return shiftSchedule;
End Task
```

Figure 28. Finding Shifts

5.5.2 Finding Shifts

The Psychic uses the generated polynomial to find the points of time where the DSSness index intersects with the *dss_threshold* or the *oltp_threshold* (see Figure 29). Therefore, the Psychic calculates the roots of the polynomial *f(t)* when $f(t) - dss\_threshold = 0$ and when $f(t) - oltp\_threshold = 0$. However, we have 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 can be easily 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.

So far, the Psychic could find all shifts that may occur but we still lack the semantics of each shift. 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 28 sketches the task of detecting shifts.

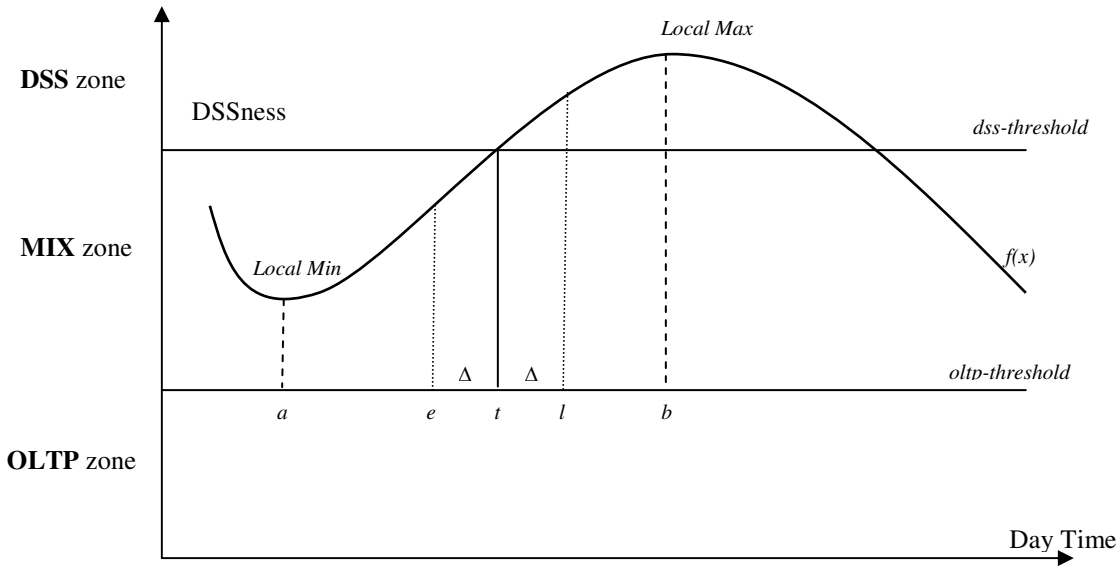


Figure 29. Shifts form when the DSSness index intersects with the thresholds.

5.5.3 Estimating Shift Check Time

This task determines the shortest period of time during which the Skeptic will run in order to validate a forecasted shift. This goal is achieved in two steps: 1) determining shift bounds, and 2) estimating earliest and latest check times.

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 29. They represent the search space, [*a*..*b*], for estimating the *earliestCheckTime* and *latestCheckTime* period, [*e*..*l*], 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 *lowerBoundCheckTime* is set to zero, which is the beginning of the day. And if the last shift is not followed by any minimum or maximum then the *upperBoundCheckTime* is set to the last minute of the day

(*max_time_scale*). Figure 30 describes how to find the [*lowerBoundCheckTime*, *upperBoundCheckTime*] period.

2) *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. Figure 31 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 on 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.

```
Task determiningShiftBounds(shiftSchedule)
 localMAndM[] = localMinimaAndMaxima();
 numLocalMAndM = localMAndM.length;

 For s =1 to shiftSchedule.numShifts
  found = false;

  For i=0 to numLocalMAndM -1
     If (localMAndM[i] > shift[s].time)

        //Handling the FIRST shift's special case
        If(i>0)
           Shift[s].lowerBoundCheckTime = localMAndM[i-1];
        Else
           Shift[s].lowerBoundCheckTime( 0);
        Endif

        Shift[s].upperBoundCheckTime = localMAndM[i];
        found = true;

        ExitLoop;
     Endif
  Endfor

  // If no min/max follows a shift
  If(NOT found)
     Shift[s].upperBoundCheckTime = MAX_TIME_SCALE;

     // Setting the lower bound
     If(numLocalMAndM > 0)
        Shift[s].lowerBoundCheckTime = localMAndM[numLocalMAndM -1];
     Else
        Shift[s].lowerBoundCheckTime = 0;
     Endif

  Endif

 Endfor
End Task
```

Figure 30. Determining Shift Bounds

```
Task estimatingEarliestAndLatestCheckTimes
   double slope;
   firstDerivative = offlinePredictionModel().derivative();

   skeptic = new Skeptic();
   For i=1 to numShifts
     shift[i].latestCheckTime= shift[i].time+ (min_check_time /2);
     shift[i].earliestCheckTime = shift[i].time - (min_check_time /2);

     // Make tests on the Consolidated Scencario
     scenario = consolidateScenarios();

     While(true)
        // Ask the Skeptic for help
        skeptic.evaluateOnlineModel(scenario, shift[i]);
        slope = skeptic.slope();

        correctTrend=((shift[i].isTrendingUp AND slope >0) OR
                      (shift[i].isTrendingDown AND slope <0));

        if (!correctTrend)
           // Expanding the checking period from left and right.
           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
           exitLoop;
         Endif

        // Stop expanding if one of the boundaries is hit.
        if (shift[i].earliestCheckTime <= shift[i].lowerBoundCheckTime OR
              shift[i].latestCheckTime >= shift[i].upperBoundCheckTime)
           exitLoop;
        Endif

      Endwhile

    Endfor // for each shift
End Task
```

Figure 31. Estimating earliest and latest check times of a shift

5.5.4 Filtering Shifts

Some of the detected shifts might not be beneficial to the performance. A shift might be too short such that it is not worth resetting 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 comparing the performance difference between the two cases:

1) The shift is accepted. This implies that the Skeptic causes some overhead due to its validation procedure, and that the DBMS's parameters are reset.
2) The shift is discarded. No validation is performed by the Skeptic, and the DBMS retains its current settings.

Figure 32 details the cost-benefit analysis used to filter shifts. Note that filtering is not needed if we have fewer than two shifts. The cost-benefit analysis is ultimately based on the global parameters *performanceMatrix* and *monCost*.

```
Task filterShifts
i=1;
If (numShifts >= 2)
 While( i< numShifts )
   shift1 = shifts[i];
   shift2 = shifts[i+1];

   if(NOT isComplement(shift1, shift2) )
     loop;

   //--- Performance if shifts are adopted
   wt = workloadTypeAt(shift1.earliestCheckTime);
   dbSettings = wt; // resetting db parameters
   c = performanceMatrix[wt, dbSettings] - monCost;
   perfIfMon = (shift1.time - shift1.earliestCheckTime)*c;

   wt = workloadTypeAt(shift1.latestCheckTime);
   perfIfMon += (shift1.latestCheckTime - shift1.time)*c;

   dbSettings = wt;
   c = performanceMatrix[wt, dbSettings];
   perfIfMon+=(shift2.earliestCheckTime - shift1.latestCheckTime)*c;

   c = performanceMatrix[wt, dbSettings] - monCost;
   perfIfMon+=(shift2.time - shift2.earliestCheckTime)*c;

   wt = workloadTypeAt(shift2.latestCheckTime);
   c = performanceMatrix[wt, dbSettings] - monCost;
   perfIfMon +=  (shift2.latestCheckTime - shift2.time)*c;


   //--- Performance if shifts are discarded
   wt = workloadTypeAt(shift1.earliestCheckTime);
   dbSettings = wt;
   c = performanceMatrix[wt, dbSettings];
   perfIfNotMon = (shift1.time - shift1.earliestCheckTime)*c;

   wt = workloadTypeAt(shift1.latestCheckTime);
   c = performanceMatrix[wt, dbSettings];
   perfIfNotMon += (shift2.time - shift1.time)* c;

   wt = workloadTypeAt(shift2.latestCheckTime);
   c = performanceMatrix[wt, dbSettings];
   perfIfNotMon += (shift2.latestCheckTime - shift2.time)*c;

   if (perfIfMon <= perfIfNotMon)
     removeShiftAt(i);
     removeShiftAt(i+1);
   End If

    i++;
  Endwhile
End Task
```

Figure 32. Filtering Shifts

```
Task setMUMParameters

  s = consolidateScenarios();

  p1 = perfWithFixedSettings(s, dominantWorkload() );
  p2 = perfUnderContinuousMonitoring(s);

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

  For i = 1 to maxUpdateDays
     setDaysToCompleteModelUpdate(i);
     p3 = perfUnderPredictionArchitecture(s);

     If (p3 > (p1+PERF_PERCENTAGE) && p3 > (p2+PERF_PERCENTAGE) )
        fastUpdateDays = i;
        fastUpdatePerf = p3;
        exitLoop;
     Endif

   Endfor // # of update days currently examined

   slowUpdateDays = maxUpdateDays;
   setDaysToCompleteModelUpdate(slowUpdateDays);
   slowUpdatePerf = perfUnderPredictionArchitecture(s);

End Task
```

Figure 33. Estimating MUM Parameters

5.5.5 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 intervals throughout the day. The MUM parameters are optimized such that the MUM satisfies the following two constraints: 1) the MUM guarantees coverage for the entire day after a *numDaysToCompleteUpdate* days, and 2) the expected performance of the prediction system is still higher than the performance of any other operation mode. To satisfy these conditions, the MUM parameters are automatically estimated by back-testing the historical data.

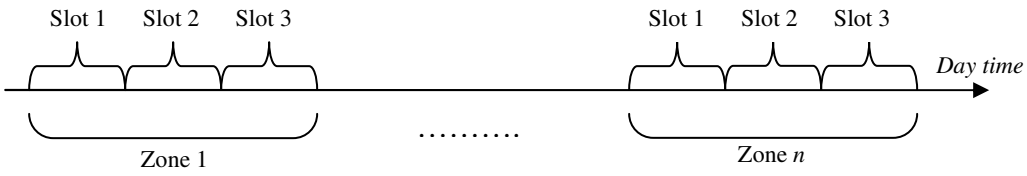


Figure 34. Regular sampling throughout the day

To understand how the MUM works, we can view the day as a number of equal, ordered time *zones*, as shown in Figure 34. Each zone is divided into a number of equal, ordered time *slots*. To ensure a uniform distribution of checked periods over the course of a day, the Skeptic runs at only one slot per zone in each day. In other words, in day one, the Skeptic is triggered at slot one of zone 1, slot 1 of zone 2, slot 1 of zone 3, etc. In day two, it runs at slot 2 of zone 1, slot 2 of zone 2, slot 2 of zone 3, etc. Therefore, if there are *n* slots in a zone, *n* days are required to make a complete coverage of a day.

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.

$$S = Day \times SlotSize + Zone \times ZoneSize$$

$$E = S + SlotSize$$

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 of each slot. *SlotSize* and *ZoneSize* are constants denoting the size of the slot and the zone respectively, where:

$$SlotSize = \min\_check\_time$$

$$ZoneSize = SlotSize \times numDaysToCompleteUpdate$$

$$numZones = \left\lceil \frac{max\_time\_scale}{ZoneSize} \right\rceil$$

We conservatively choose to set *SlotSize* to *min_check_time*, which is the shortest check period that the Skeptic uses to validate a shift. *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 *fastUpdateDays* to *slowUpdateDays*, where *fastUpdateDays* denotes the minimum number of days that the MUM needs in order to complete the update while the performance of the architecture remains superior, and *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 $slowUpdateDays = \left\lceil \frac{max\_time\_scale}{SlotSize} \right\rceil$. *fastUpdateDays* is optimized by starting by one day and incrementing it until the prediction architecture outperforms the other operation modes (Section 5.7) by an arbitrary percentage PERF_PERCENTAGE, that is, *fastUpdatePerf* >= (*anyOtherPerf*+PERF_PERCENTAGE). We use PERF_PERCENTAGE=1% in our experiments. *fastUpdatePerf* and *slowUpdatePerf* are the performance measures associated with setting *numDaysToCompleteUpdate* to *fastUpdateDays* and *slowUpdateDays* respectively. Therefore, *numDaysToCompleteUpdate* creates a tradeoff between the pace at which the architecture can fully update a training scenario and the performance level. Setting *numDaysToCompleteUpdate* to *fastUpdateDays* leads to a faster update, but with a relatively lower performance due to the incurred run-time monitoring. Setting *numDaysToCompleteUpdate* to *slowUpdateDays* leads to the maximum DBMS

Variable	Description
numDaysToCompleteUpdate	Number of days needed to update an entire scenario
Slot	The slot number in a particular zone.
Zone	The zone number within the day
Day	The day number since the start of the MUM update
S	The next start time for the MUM to work
E	The end time for the current update session
slotSize	Size of the slot
zoneSize	Size of the zone
numZones	Number of zones in a day
fastUpdateDays	Minimum # of days to update an entire scenario
slowUpdateDays	Maximum # of days to update an entire scenario
fastUpdatePerf	Overall performance using *fastUpdateDays*
slowUpdatePerf	Overall performance using *slowUpdateDays*

Table 14. Variables used in the Model Update Mechanism (MUM)

performance but a longer time is required to update the model. In general, the architecture has the ability to estimate the DBMS's performance for any value assigned to *numDaysToCompleteUpdate* and vice versa. Figure 33 shows how to estimate the MUM parameters that were discussed. Table 14 summarizes the variables used in this estimation.

5.6 THE SKEPTIC

The Skeptic's main function is to validate the Psychic's forecasted shifts. For each upcoming shift, the Skeptic samples the workload from *earliestCheckTime* to *latestCheckTime*. The workload samples are analyzed to confirm whether the trend of a shift conforms to the Psychic's prediction. 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. 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-MUM collaboration. Whether the MUM is enabled or disabled, the Skeptic samples collected during the validation process are deemed the most recent observations of the day during that interval. Therefore, in addition to the regular sampling performed by the MUM mechanism, the Skeptic also passes its own samples collected throughout the validation task to the MUM component in order to patch the historical data stored in the TrainingDataModel.

5.7 OPERATION MODES

A DBMS can run in one of the following operation modes: out-of-the-box (default) settings, fixed settings suitable for the dominant workload, dynamic settings using continuous monitoring, or the Psychic-Skeptic architecture. Our experiments compare the performance of the DBMS under these different modes and show that the Psychic-Skeptic architecture has the potential to outperform the other modes. Next, we briefly describe how the DBMS handles the workload under each operation mode.

5.7.1 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.

5.7.2 Dominant Workload Mode

In 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.

5.7.3 Continuous Monitoring Mode

This is supposed to be the most adaptable mode that can fully take advantage of the Workload Classifier. It performs on-line, short-term prediction using the moving average (MA) [63]:

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

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 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 just check before the expected start time of the shift, and therefore, we use half of the interval.

This 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. Therefore, it is hard to speculate on the performance of this mode with respect to the two above without back-testing the historical data. In general, this mode is very adaptive but costly.

5.7.4 Psychic-Skeptic Mode

Assuming the existence of a repeatable pattern in the workload, we argue that the overall performance of the system can be superior using the Psychic-Skeptic prediction system as shown in the following experiments.

5.8 EXPERIMENTS

Our experiments have two main goals. First, we validate the Psychic-Skeptic approach and compare its performance to the alternative operation modes. Second, we show that our approach is robust and able to adapt to changes that may occur in the workload pattern.

We test our architecture using artificially generated data that allow us to examine specific cases as well as arbitrary situations. DSSness data are generated using the notion of *Scenarios* and *ScenarioDescriptors*. A *ScenarioDescriptor* can be perceived as a template for generating scenarios that exhibit a particular daily pattern. Therefore, a particular *ScenarioDescriptor* is used as a factory to generate multiple daily scenarios that exhibit a particular pattern. A *ScenarioDescriptor* can represent any workload pattern that may characterize a special event or season (e.g., statuary holidays, Christmas shopping days, weekends, weekdays, etc.) over any window of time (day, week, etc.).

A *ScenarioDescriptor* consists of a set of pairs (*time*, *DSSness*) that play the role of *anchors* of DSSness values on the final DSSness curve. The *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 $\pm$(0-5)% of random noise in the DSSness, and $\pm$(0-2)% of random noise in the time, which is equivalent to $\pm$(0-30) minutes. Noise injected in the time dimension affects when a shift may start or end. Noise injected in the DSSness dimension affect whether a shift is likely to occur or not based on its intersection with the threshold lines.

In each experiment, we simulate[13] the performance of a DBMS run under each of the four operation modes based on the empirically-obtained parameters described in Section 5.3.1. The performance of the Psychic-Skeptic architecture is evaluated when the MUM

[13] The Psychic-Skeptic Architecture is implemented in Java.

is enabled and when it is disabled. We use the default parameter settings illustrated in Table 11, unless otherwise indicated.

In these experiments we report the following:

- The expected performance using the architecture with the MUM enabled.
- The minimum and maximum performance expected from the architecture under fast and slow MUM, respectively.
- The performance of the DBMS under each of the four operation modes.
- The relative performance improvements and degradation of the various operation modes with respect to the Default Mode.

5.8.1 Experiment 1: Pattern A

The goal of this experiment is to examine the performance of the DBMS under the pattern A generated by the *ScenarioDescriptor* = { (0, 10), (150, 20), (160,23), (250, 47), (350, 60), (450, 70), (470, 67), (500, 63), (650, 58), (660, 50), (850, 25), (900, 20), (1000, 17), (1050, 15), (1150, 16), (1200, 17), (1440, 50)}. Figure 35 shows an instance scenario of this daily pattern. As seen, the DBMS experiences a workload that is mostly OLTP in the first two hours of the day. Then it changes to a mixed workload over the next 12 hours. In the next 9 hours, the dominant workload becomes OLTP, and then it 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.

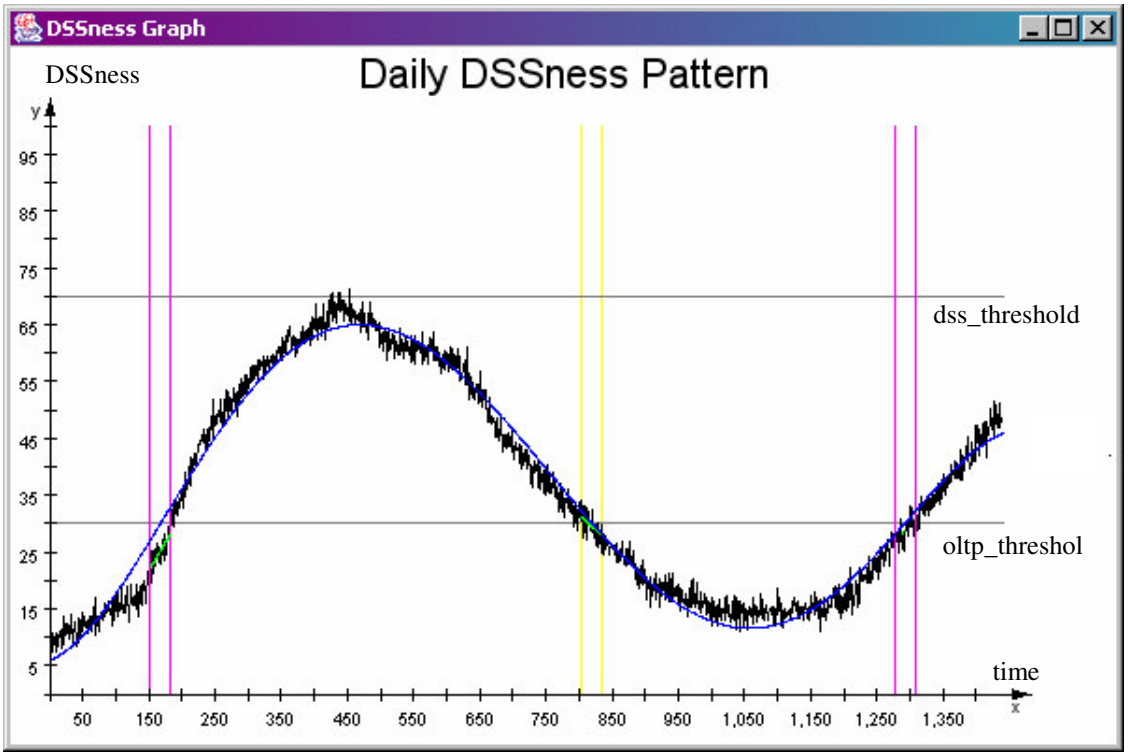


Figure 35. An example of the daily pattern A.

	Shift Type	EarliestCheckTime	Time	LatestCheckTime
Shift 1	OLTP_UP_TO_MIX	152	167	182
Shift 2	MIX_DOWN_TO_OLTP	804	819	834
Shift 3	OLTP_UP_TO_MIX	1278	1293	1308

Table 15: Shifts of Pattern A

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

$$f(x) = -6.61 + 0.32*X - 4.07E\text{-}4*X^2 + 2.64E\text{-}8*X^3 + 8.73E\text{-}11*X^4$$

The initial shift schedule, which may change later if the MUM is enabled, for this daily pattern is shown in Table 15. Table 16 summarizes the performance statistics observed with the Pattern A while MUM is ON and OFF, using the four operation modes. Note that by the absolute performance we refer to the performance percentage that can be achieved with respect to the maximum, theoretical performance resulting from matching the DBMS settings with the workload type for each minute. By the relative performance we refer to the percentage of performance improvement (or degradation) of any operation mode with respect to the Default Mode.

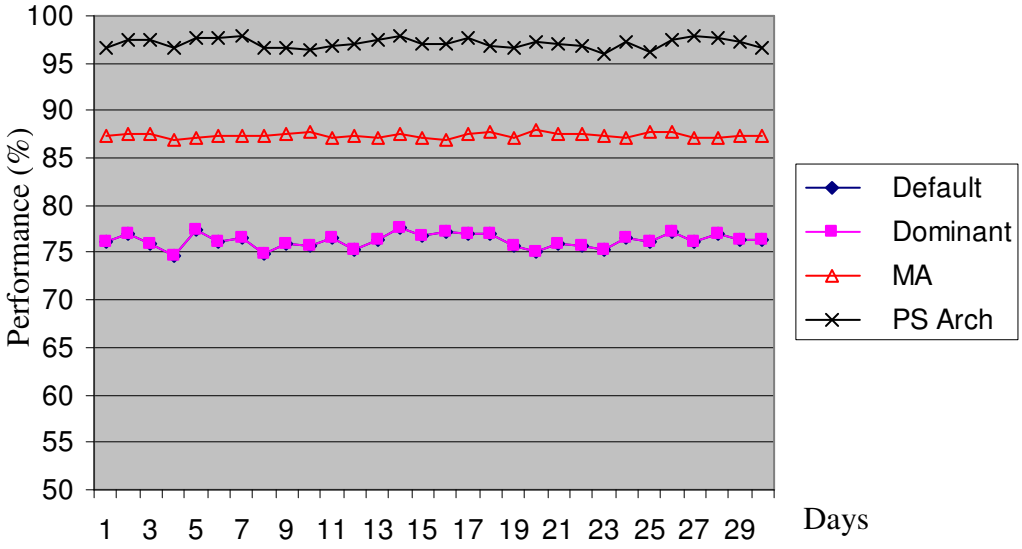


Figure 36. Absolute performance of pattern A (MUM is off).

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

Table 16: The DBMS's performance under pattern A

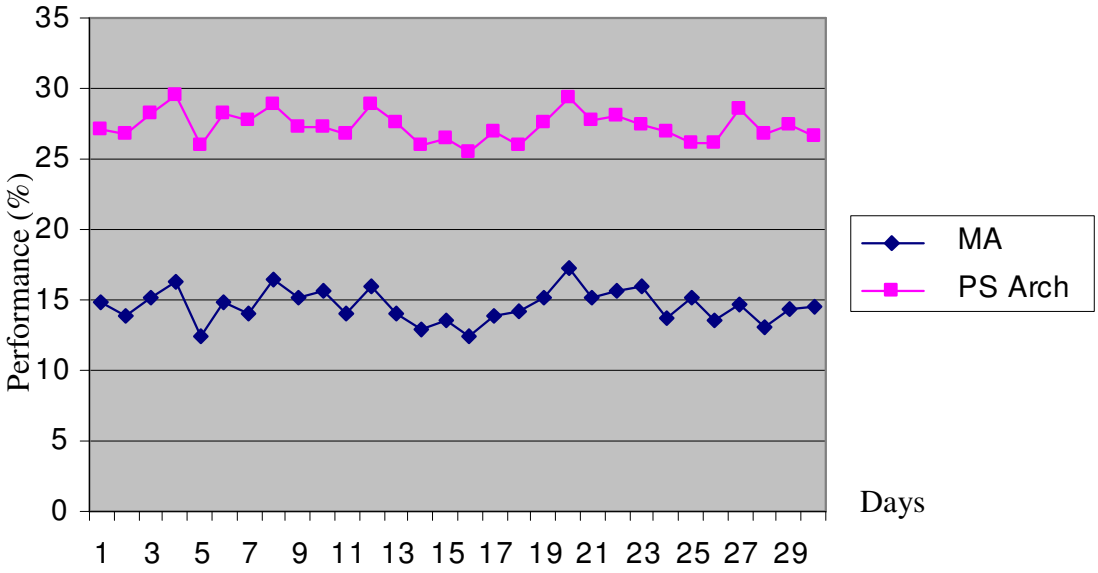


Figure 37. Relative performance of pattern A (MUM is off).

We should notice also that the mixed workload is dominant for pattern A. Therefore, the performance obtained under the Dominant Mode is equivalent to the Default Mode.

Performance when MUM is OFF. All modes are tested for at least 30 days with the MUM turned off. Figure 36 shows the DBMS's absolute performance under different operation modes. The best performance is achieved under the Psychic-Skeptic architecture (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 37 shows that the Psychic-Skeptic architecture 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% (*monCost* = 10%).

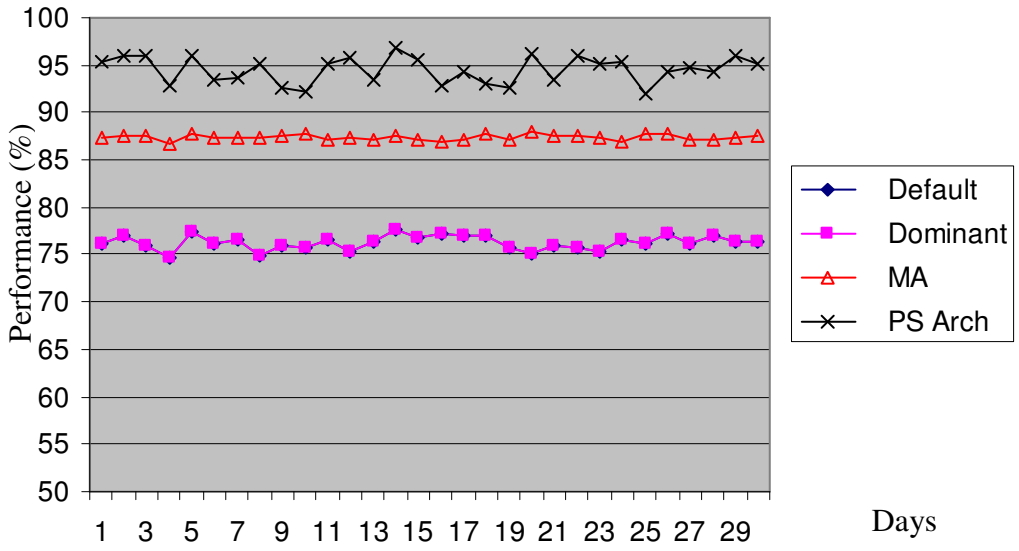


Figure 38. Absolute performance of pattern A (MUM is ON).

Performance when MUM is ON. When the MUM is enabled, the architecture reports the following estimates:

- The minimum number of days (*fastUpdateDays*) needed to complete updating the most recent scenario in the *TraningDataModel*. It also reports the expected performance (*fastUpdatePerf*) if the system uses *fastUpdateDays* to complete the update.
- The maximum number of days (*slowUpdateDays*) needed to complete the model update, and the associated (*slowUpdatePerf*).

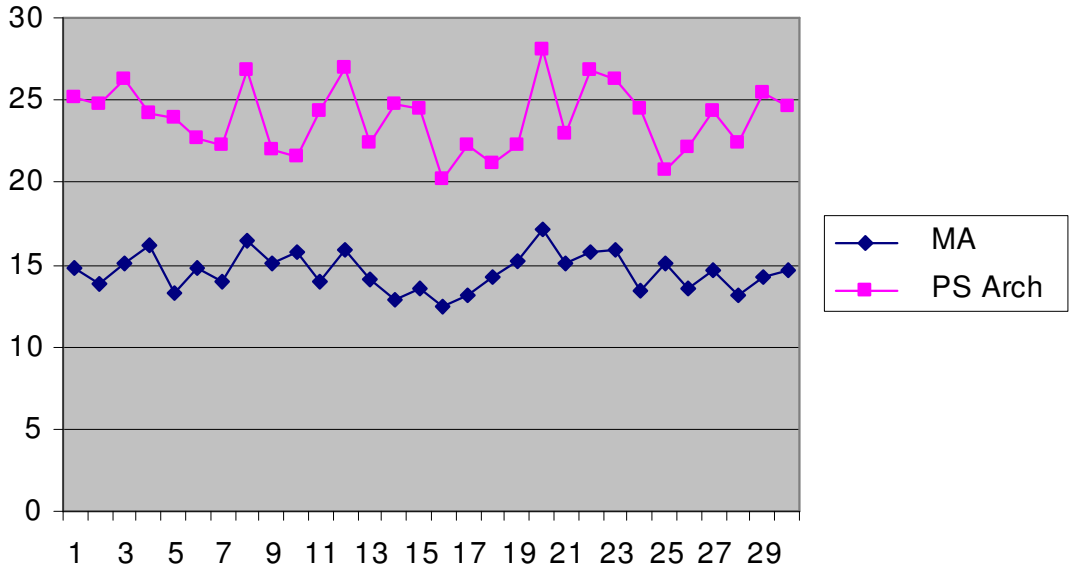


Figure 39. Relative performance of pattern A (MUM is ON).

	Days	Performance
Fast Upate	*fastUpdateDays* = 2	*fastUpdatePerf* = 92.78%
Slow Update	*slowUpdateDays* = 48	*slowUpdatePerf* = 97.58%

Table 17. Min and Max Performance of Pattern A while MUM is ON

Setting the *numDaysToCompleteUpdate* parameter to the minimum number of days for the update, *fastUpdateDays*, results in a system that is updated quickly while retaining superior performance overall other operation modes. On the other extreme, the business may decide to minimize overhead by setting *numDaysToCompleteUpdate* to the maximum number of updating days, *slowUpdateDays*. Therefore, *numDaysToCompleteUpdate* creates a tradeoff between the pace of model update and overall performance. Notice that the architecture can estimate the performance under any given *numDaysToCompleteUpdate*, and vice versa. Knowing the expected performance beforehand allows the DBMS to issue an alert in case the performance is degrading. Table 17 indicates the minimum performance expected when

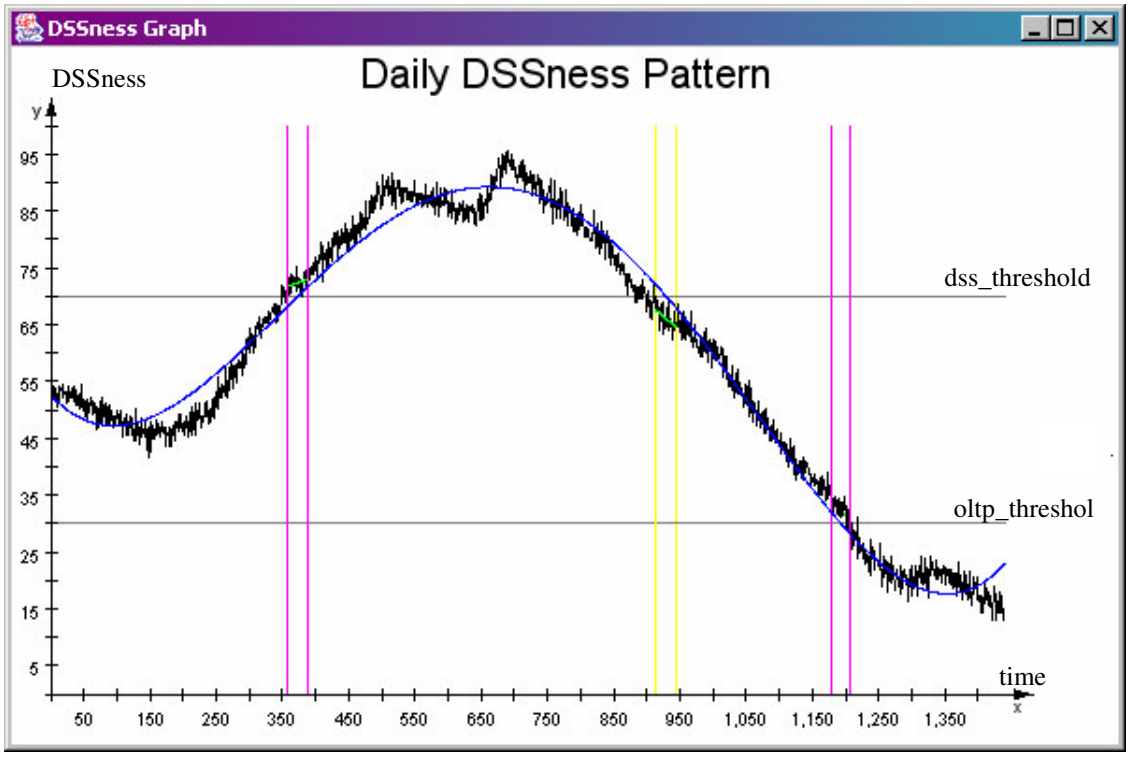


Figure 40. An example of the daily pattern B.

numDaysToCompleteUpdate is set to *fastUpdateDays*. It also shows the maximum performance when *numDaysToCompleteUpdate* is set to *slowUpdateDays*.

Figure 38 compares the performance of the DBMS under the four operation modes. Regarding the Psychic-Skeptic architecture, the MUM is turned on with *numDaysToCompleteUpdate =7*, that is, a complete daily scenario will be updated after one week. The Psychic-Skeptic architecture still outperforms others (avg. 94.54%), followed by the MA Mode (avg. 87.37%), then the Dominant Mode (76.23%), which is equivalent to Default Mode. Figure 39 shows that the Psychic-Skeptic architecture achieves an average of 24.03% performance improvement over the Default Mode compared to 14.63% achieved by the MA Mode.

MUM	Perf.	Statistic	Default	Dominant	MA	PS Arch
OFF	Absolute	Mean	74.04	74.04	87.74	95.95
		Std. Dev.	0.91	0.91	0.25	1.42
		Conf. Int.	[73.73, 74.35]	[73.73, 74.35]	[87.66, 87.83]	[95.46,96.44]
	Relative	Mean	n/a	0	18.53	29.62
		Std. Dev.	n/a	0	1.37	2.83
		Conf. Int.	n/a	n/a	[18.06, 19.01]	[28.64, 30.60]
ON	Absolute	Mean	74.04	74.04	87.74	94.19
		Std. Dev.	0.91	0.91	0.26	1.47
		Conf. Int.	[73.73, 74.35]	[73.73, 74.35]	[87.63, 87.82]	[93.68, 94.70]
	Relative	Mean	n/a	0	18.50	27.24
		Std. Dev.	n/a	0	1.37	2.72
		Conf. Int.	n/a	n/a	[18.03, 18.97]	[26.30, 28.18]

Table 18. The DBMS's performance under pattern B

5.8.2 Experiment 2: Pattern B

The goal of this experiment is to examine the performance of the DBMS under the pattern B generated by the *ScenarioDescriptor* = { (0, 55), (150, 45), (160, 46), (250, 50), (350, 70), (450, 80), (500, 90), (650, 85), (700, 95), (850, 80), (900, 70), (1000, 60), (1050 55), (1150, 40), (1200, 35), (1250, 25), (1300, 20), (1350, 23), (1440, 15)}. Figure 40 shows an instance scenario of this daily pattern. As seen, the DBMS experiences a workload that is mostly mixed in the first 8 hours of the day. Then it starts to be 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.

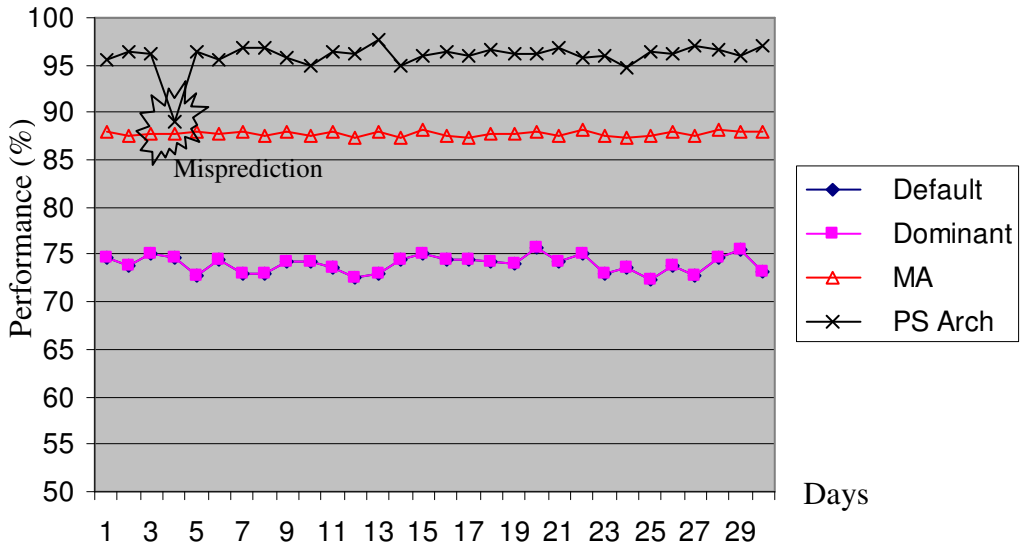


Figure 41. Absolute performance of pattern B (MUM is off).

	Shift Type	EarliestCheckTime	Time	LatestCheckTime
Shift 1	MIX_UP_TO_DSS	358	373	388
Shift 2	DSS_DOWN_TO_MIX	914	929	944
Shift 3	MIX_DOWN_TO_OLTP	1178	1193	1208

Table 19. Shifts of Pattern B

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

$$f(x) = 52.53 - 0.12*X + 7.71E\text{-}4*X^2 - 9.98E\text{-}7*X^3 + 3.55E\text{-}10*X^4$$

The initial shift schedule, which may change later if the MUM is enabled, for this daily pattern is shown in Table 19. Table 18 summarizes the performance statistics observed with Pattern B while MUM is ON and OFF, using the four operation modes.

The dominant workload for pattern B is MIX, therefore, the performance under the Dominant and the Default modes is equivalent.

Performance when MUM is OFF. All modes are tested for at least 30 days with the MUM turned off. Figure 41 shows the DBMS's absolute performance under the different

	Days	Performance
Fast Update	*fastUpdateDays*= 2	*fastUpdatePerf* = 91.34%
Slow Update	*slowUpdateDays* = 48	*slowUpdatePerf* = 95.91%

Table 20. Min and Max Performance of Pattern B while MUM is ON

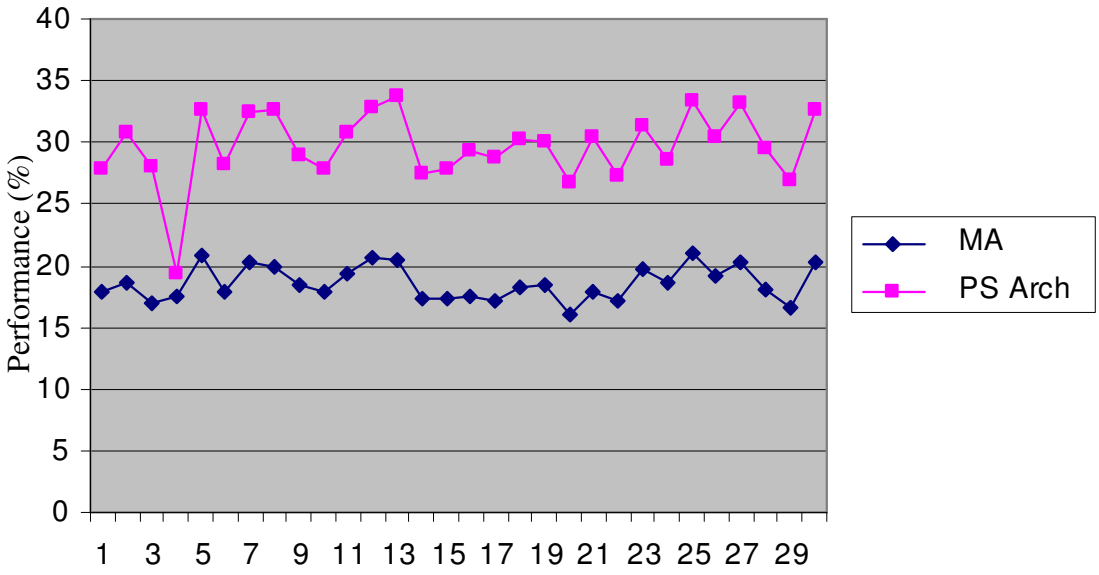


Figure 42. Relative performance of pattern B (MUM is off).

operation modes. The best performance is achieved under the Psychic-Skeptic architecture (avg. 95.95%), followed by the MA Mode (avg. 87.74%), then the Dominant Mode (avg. 74.04.23%), which is equivalent to the Default Mode. Figure 42 shows that the Psychic-Skeptic architecture achieved an average of 27.24% performance improvement over the Default Mode compared to 18.50% achieved by the MA Mode. All performance estimates are based on workload classifier overhead of 10% (*monCost*= 10%). The Skeptic invalidated the shift that is supposed to occur at minute 1193 in the fourth day. This misprediction causes remarkable performance degradation in that day as shown in Figure 41. Another shift invalidation occurs at minute 929 in the 15th day but it has insignificant impact on the performance.

Performance when MUM is ON. Table 20 indicates the minimum performance expected when *numDaysToCompleteUpdate* is set to *fastUpdateDays*. It also shows the maximum performance when *numDaysToCompleteUpdate* is set to *slowUpdateDays*.

Figure 43 compares the performance of the DBMS under the four operation modes. Regarding the Psychic-Skeptic architecture, the MUM is turned on with *numDaysToCompleteUpdate =7*, that is, a complete daily scenario is updated after one week. The Psychic-Skeptic architecture 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. Figure 44 shows that the Psychic-Skeptic architecture achieves an average of 27.24% performance improvement 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 20^{th} days. These mispredicted shifts cause no severe impact on the performance. However, the Skeptic invalidates the shift that is supposed to occur at minute 1201 in the 9^{th} day. As seen in Figure 43, this misprediction has a significant impact on the overall performance in that day.

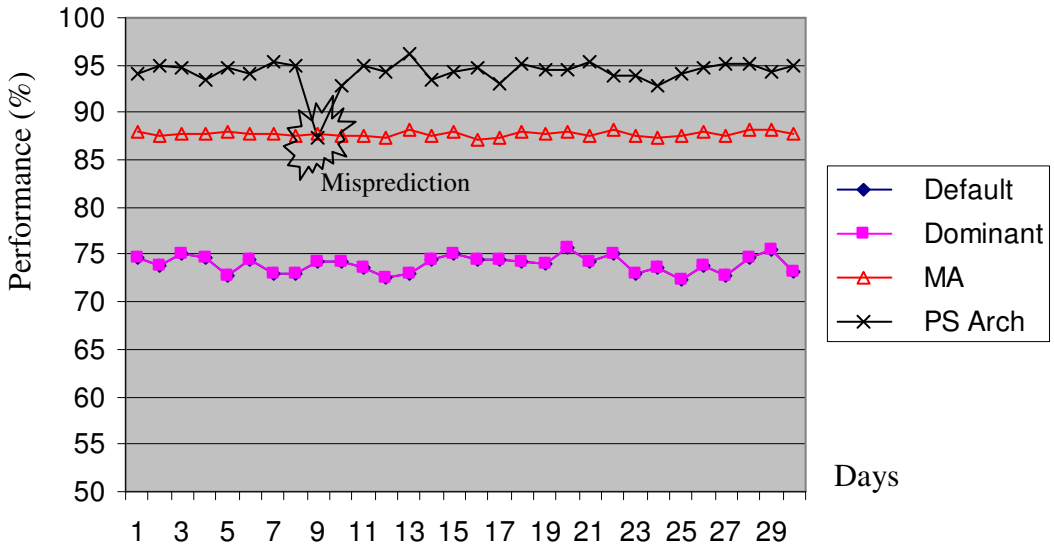


Figure 43. Absolute performance of pattern B (MUM is ON).

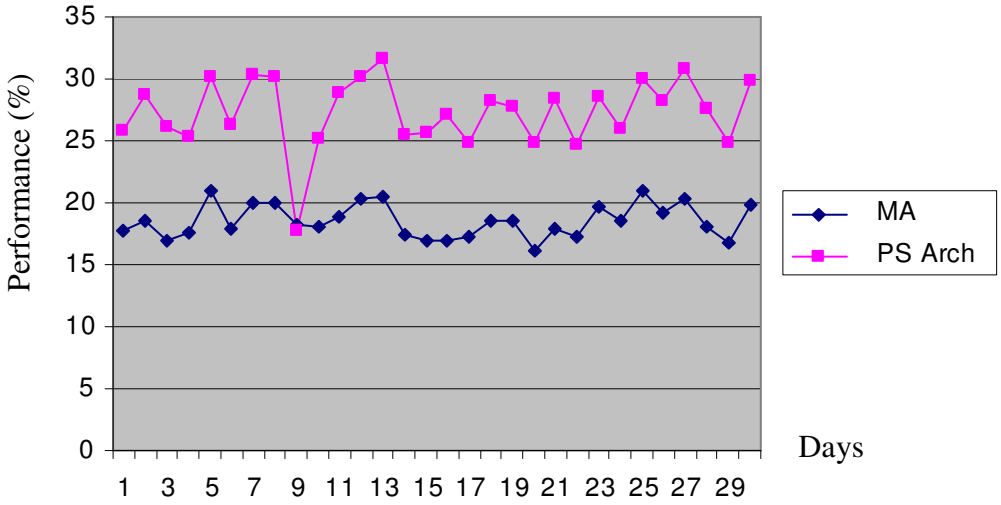


Figure 44. Relative performance of pattern B (MUM is ON).

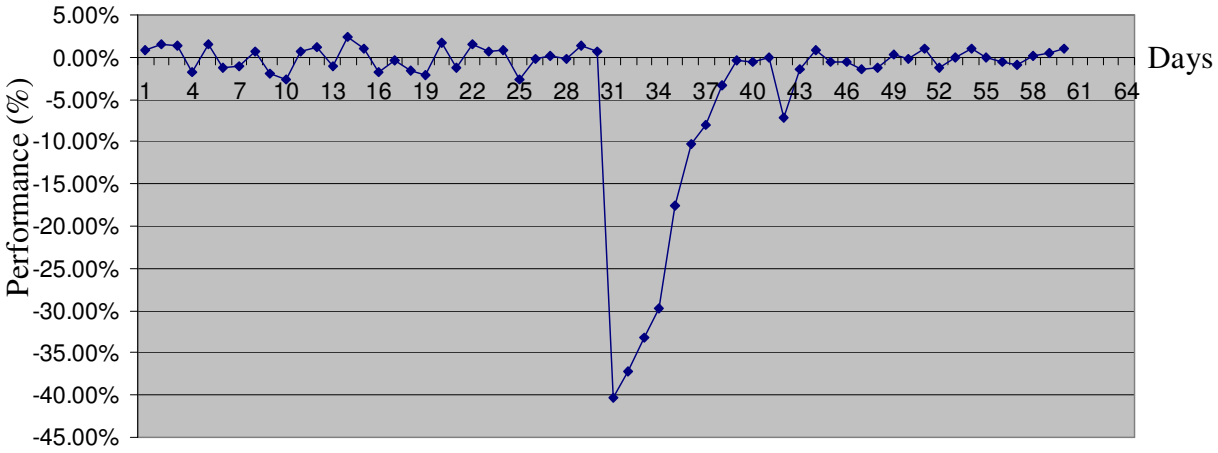


Figure 45. Adaptability test: Transition from pattern A to pattern B

5.8.3 Adaptability: Pattern A changes to Pattern B

The goal of this experiment is to demonstrate one of the vital features of the Psychic-Skeptic architecture, which is its adaptability to changes in the daily pattern. We run the DBMS under pattern A, described above, for 30 days. Then, we swiftly switch to pattern B, under which the behavior of the DBMS is examined for another 30 days. This sudden shift in the daily pattern is unrealistic as changes usually happen gradually over several days. However, this swift change allows us to aggressively push the architecture 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. Therefore, in this experiment we 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:

$$perfDiff = \frac{(Observed - Expected)}{Expected} \times 100$$

Figure 45 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, 94.50%, as compared to the expected performance of 94.54%. At the end of the first 30 days processing pattern A, we exposed the DBMS to pattern B. On day 31, the performance dramatically degrades as it drops to 56.3% (that is, a 40% degradation) from the expected performance (94.19%). However, since the MUM is enabled, the performance gradually catches up 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.

5.9 SUMMARY

Monitoring systems and constant on-line analysis cause performance penalties that may hinder the adaptation of tools such as the workload classifier. Luckily, the overhead of such tools can be mitigated by exploiting characteristics in the workload. In this chapter, we introduced the Psychic-Skeptic prediction architecture. 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 put full trust in the off-line predicted shifts. Therefore it 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 architecture is self-optimizing as the majority of its parameters is automatically estimated and is transparent to the end user (DBA). The architecture adapts to changes in the workload pattern as it is equipped by the Model-Update-Mechanism (MUM) that samples the workload at regular times in order to patch historical data and keep prediction models up to date.

Our experiments have two goals. The first goal is to assess the performance of the DBMS using the Psychic-Skeptic architecture. Experiments show that the prediction architecture outperforms other modes of operation, namely the Default Mode, the Dominant Workload Mode, and the Constant Monitoring using the MA. The second goal is to demonstrate the adaptability of the architecture. Our experiments show that the architecture is robust against changes in the workload pattern. Although the pattern transition we experimented with was swift and stiff, the architecture managed to learn the new pattern within the expected period of time. Therefore, we consider the MUM a self-healing mechanism as 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. Some of these scenarios are shown in Appendix A.

CHAPTER 6 CONCLUSIONS

Database Management Systems (DBMSs) are complex systems whose manageability is increasingly becoming a real concern. Realizing that there is a dearth of skilled DBAs and that the cost of hiring them is a major part of the Total Cost of Ownership (TCO) makes Autonomic DBMS (ADBMS) indispensable.

In this thesis, we discussed the important characteristics that DBMSs should have in order to be self-managing. A closer look at present, prominent commercial DBMSs reveals that there is a lot of effort remaining to make them autonomous. The complexity of managing these systems in particular stem from several sources such as the incrased emphasis on QoS, the numerous functionalities and advanced features added everyday, housekeeping tasks, expanding database size, and the strong trending towards e-service era. We pointed out to several research areas that need attention in order to mitigate the complexity of management DBMSs.

We also stressed one of the most imperative properties of ADBMSs, which is being workload-aware. Therefore, we studied the various workload characterization techniques used in different computing areas. Such a study helped us determine what technique to choose in order to explore interesting properties and patterns in the DBMS workload. As a proof of concept, we showed how the type of the workload, specifically whether it OLTP or DSS, is a key criterion for performance optimization. We developed a methodology by which the DBMS can automatically recognize its workload type and assess its concentration in the overall workload mix. This methodology is primarily based on data mining classification techniques. We demonstrated the success of this methodology by using artificial and real workloads. This piece of work shows the utility

of exploiting the static characteristics of the workload. We then presented the prediction architecture that analyzes the dynamic characteristics of the workload in order to forecast its change over time. This prediction architecture is adaptive and generic such that it can be used to solve other similar prediction problems.

6.1 OUR POSITION IN THE AUTONOMIC PATH

To implement autonomic computing features, we advocate the evolutionary approach that delivers improvements to current systems in order to provide significant self-managing value to the end users without requiring them to completely replace their current IT environments. Five evolutionary levels have been envisioned in order to reach fully autonomic systems [33]. Figure 46 is a representation of those levels, starting from the *basic* level, through *managed*, *predictive*, and *adaptive* levels, and finally to the *autonomic* level.

As seen in the figure, the *basic* level represents the starting point where some IT systems are today. Each system element is managed independently by IT professionals who set it up, monitor it, and eventually replace it. At the *managed* level, systems management technologies can be used to collect information from disparate systems onto fewer consoles, reducing the time it takes for the administrator to collect and synthesize information as the systems become more complex to operate. In the *predictive* level, as new technologies are introduced that provide correlation among several elements of the system, the system itself can begin to recognize patterns, predict the optimal configuration, and provide advice on what course of action the administrator should take. As these technologies improve, and as people become more comfortable with the advice

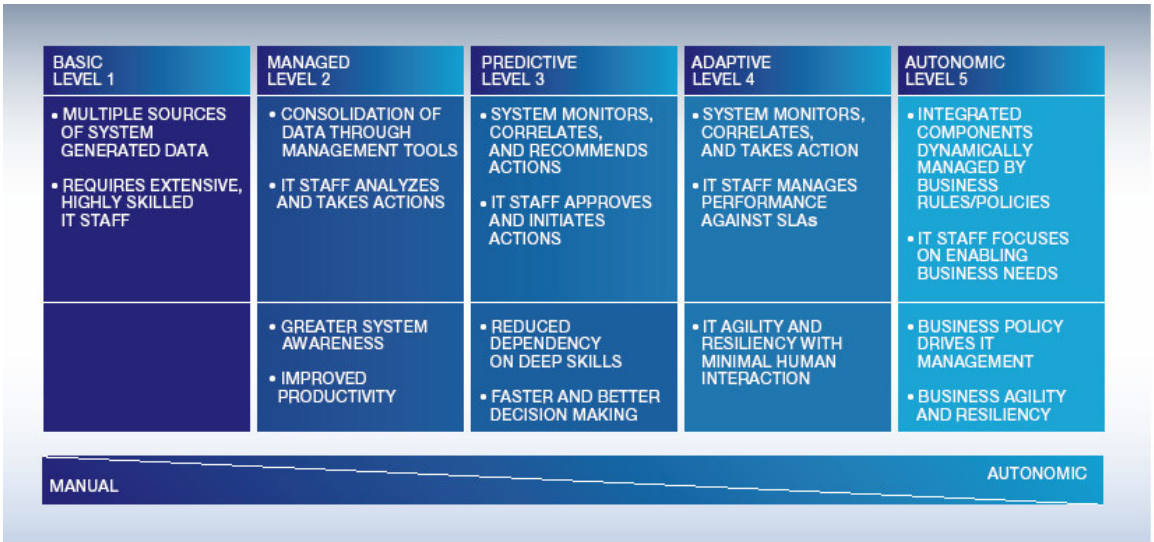


Figure 46. Evolution not revolution [33]

and predictive power of these systems, we can progress to the *adaptive* level where the systems themselves become self-learners and can automatically take the correct actions based on the information that is available to them and the knowledge of what is happening in the systems. Finally, at the fully *autonomic* level, the system operation is governed by business policies and high level objectives. Users interact with the system to monitor the business processes or alter the objectives.

We consider our work in this thesis goes inline with this progressive path towards having autonomic DBMSs as it fits mostly in the *predictive* and *adaptive* levels described above.

6.2 RESEARCH PLANS

Our thesis poses a number of future research directions such as:

- Investigating the feasibility of using the Psychic-Skeptic architecture 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.

- Developing a *Model Validation Mechanism*, with which a workload classifier can automatically validate itself with respect to drastic changes in the properties of the business's workload. The system will therefore be able to determine when to refresh the model in order to maintain high classification accuracy.
- Investigating the feasibility of adopting new database architectures (e.g., RISC-like architecture) that may reduce the complexity of managing their performance and ease their integration with other systems.
- Using control theory and fuzzy logic to control complex systems like DBMSs. These concepts have been used [41] to manage what might be less intricate systems than DBMSs so it is worth investigating their feasibility with respect to databases.

REFERENCES

[1] Aberdeen Group. “Database Cost of Ownership Study,” http://relay.bvk.co.yu/progress/aberdeen/aberdeen.htm, 1998.

[2] Aberdeen Group. *IBM Data Management Tools: New Opportunities for Cost-Effective Administration*, Profile Report, Aberdeen Group, Inc., Boston (April 2002), p. 3.

[3] Aboulnaga, A. and Chaudhuri, S. "Self-tuning Histograms: Building Histograms Without Looking at Data". *Proceedings of ACM SIGMOD*, Philadelphia, 1999.

[4] Agrawal, S., Chaudhuri, S., and Narasayya, V. "Automated Selection of Materialized Views and Indexes for SQL Databases". *VLDB 2000*, pp. 496-505.

[5] Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y. "Hippocratic Databases". *VLDB 2002*, Hong Kong, China. VLDB Endowment, (2002).

[6] Ailamaki, A., DeWitt, D., Hill, M., and Wood, D. “DBMSs On A Modern Processor: Where Does Time Go?,” *Proc. of Int. Conf. On Very Large Data Bases (VLDB ‘99)*, 266-277, (Sept 1999).

[7] Barroso, L., Gharachorloo, K., and Bugnion, E. "Memory System Characterization of Commercial Workloads" *Proc. Of the 25th International Symposium on Computer Architecture*, 3-14, (June 1998).

[8] Bernstein, P. "Applying Model Management to Classical Meta Data Problems" *Proceedings of the first Biennial Conference on Innovative Data Systems Research*, VLDB, (January 2003).

[9] Brown, A. and Patterson, D. “To Err Is Human,” *Proceedings of the First Workshop on Evaluating and Architecting System Dependability (EASY ’01)*, Goeteborg, Sweden (July 2001).

[10] Calzarossa, M. and Serazzi, G.. "Workload Characterization: a Survey". *Proceedings of the IEEE* 81, 8, 1136-1150, (August 1993).

[11] Castano, S., Fugini , M., Martella, G., and Samarati , P. *Database Security*. AddisonWesley, (1995).

[12] Calzarossa, M. and Ferrari, D. "A Sensitivity Study of the Clustering Approach to Workload Modeling". *Performance Evaluation* 6, 1, 25-33, (1986).

[13] Calzarossa, M., Massari, L. and Tessera, D. "Workload Characterization – Issues and Methodologies". *Performance Evaluation - Origins and Directions,* volume 1769 of *Lecture Notes in Computer Science,* Haring, G., Lindemann, C. and Reiser, M., Eds., Springer-Verlag, 459-484.

[14] Calzarossa, M. and Serazzi, G. "A Characterization of the Variation in Time of Workload Arrival Patterns". *IEEE Trans. On Computers* 34, 2, 156-162., (1985).

[15] Calzarossa, M. and Serazzi, G. "Construction and Use of Multiclass Workload Models". *Performance Evaluation* 9, 4, 341-352, (1994).

[16] Chaudhuri, S., and Dayal, U. "An Overview of Data Warehousing and OLAP Technology". *SIGMOD Record* 26*,* 1 (March 1997).

[17] Chaudhuri, S. and Narasayya, V. "Automating Statistics Management for Query Optimizers". *Proceedings of 16th International Conference on Data Engineering*, San Diego, USA (2000).

[18] Chaudhuri, S. and Weikum, G. “Rethinking Database Architecture,” *Proceedings of 26th VLDV,* Cairo, Egypt, (2000).

[19] Crovella, M. and Lindemann, C. Internet Performance Modeling: The State of the Art at the Turn of the Century. *Performance Evaluation* 42, 91-108, (2000).

[20] Cunha, C., Bestavros, A., and Crovella, M. "Characteristics of WWW Client-based Traces". *Technical Report BU-CS-95-010,* Computer Science Dept., Boston University, (1995).

[21] Dan, A., Yu, P., and Chung, J. “Characterization of Database Access Pattern for Analytic Prediction of Buffer Hit Probability,” *Very Large Data Bases (VLDB) Journal* **4**, No. 1, 127-154, (1995).

[22] D.H. Brown Associates. “DB2 UDB vs. Oracle8i: Total Cost of Ownership”. D.H. Brown Associates, Inc., Port Chester, NY.. http://www.breakthroughdb2.com/., (December 2000).

[23] Elms, C. "Clustering - One method for Workload Characterization". *Proceedings of the International Conference on Computer Capacity Management*, San Francisco, Calif., (1980).

[24] Elnaffar, S. "A Methodology for Auto-Recognizing DBMS Workloads". *Proceedings of Centre for Advanced Studies Conference (CASCON ’02)*, (October 2002).

[25] Elnaffar, S., and Martin, P. "Characterizing Computer Systems’ Workloads". *Technical Report 2002-461*, School of Computing, Queen’s University, Canada, (Dec. 2002).

[26] Elnaffar, S., Martin, P., and Horman, R. "Automatically Classifying Database Workloads". *Proceedings of ACM Conference on Information and Knowledge Management (CIKM ACM '02)*, (November 2002).

[27] Elnaffar, S., Powley, W., Benoit, D. and Martin, P. "Today's DBMSs: How Autonomic Are They?" *Proceedings of the First IEEE International Autonomic Systems Workshop*, DEXA 2003, Prague, (2003).

[28] Elnaffar, S., Powley, W., Benoit, D. and Martin, P. "Today's DBMSs: How Autonomic Are They?" *Technical Report 2003-469,* School of Computing, Queen's University, Canada, (Sept. 2003).

[29] Evans-Correia, K. "Simplifying Storage Management Starts with More Efficient System Utilization," Interview with N. Tabellion, searchStorage (August 2001), see *http://searchstorage.techtarget.com/qna/0,289202,sid5_gci764063,00.html*.

[30] Ferrari, D. "On the Foundations of Artificial Workload Design". *Proc. of ACM Sigmetrics Conf. On Measurement and Modeling of Computer systems*, Cambridge, MA, 8-14.

[31] Ferrari, D., Serazzi, G., and Zeigner, A. *Measurement and Tuning of Computer Systems*, Prentice Hall, Englewood Cliffs, N.J., (1983).

[32] Fu, K. *Syntactic Methods in Pattern Recognition*, Academic Press.

[33] Ganek, A. and Corbi, T. "The Dawning of the Autonomic Computing Era". *IBM Systems Journal*, 42, 1, (March 2003).

[34] Gassner, P., Lohman, G., Schiefer, B., and Wang, Y. "Query Optimization in the IBM DB2 Family". *IEEE Data Engineering Bulletin*, 16(4), pp. 4-18, (1993).

[35] Golding, R, Bosch, P., Staelin, C., Sullivan, T. and Wilkes, J. "Idleness is not sloth." *Proceedings of the Winter Usenix Conference*, New Orleans, Louisiana, pp. 201-12, (January 1995).

[36] Han, J., Chiang, J., Chee, S., Chen, J., Chen, Q., Cheng, S., Gong, W., Kamber, M., Liu, G., Koperski, K., Lu Y., Stefanovic, N., Winstone, L., Xia, B., Zaiane, O., Zhang, S. and Zhu, H. 1997. DBMiner: A System for Data Mining in Relational Databases and Data Warehouses. In *Proc. CASCON '97: Meeting of Minds,* Toronto, Canada, 249-260, (November 1997).

[37] Harizopoulos, S. and Ailamaki, A. "A Case for Staged Database Systems" *Proceedings of the first Biennial Conference on Innovative Data Systems Research*, VLDB, (January 2003).

[38] Howard, R. *Dynamic Programming and Markov Processes*, John Wiley.

[39] Harman, H.. *Modern Factor Analysis*, University of Chicago Press, Chicago, IL.

[40] Hartigan, J. and Wong, M. "A K-means Clustering Algorithms". *Applied Statistics* 28, 100-108.

[41] Hellerstein, JL and Parekh, S. "An Introduction to Control Theory with Applications to Computer Science," *ACM Sigmetrics*, (2001).

[42] Hofmann, R., Klar, R., Mohr, B., Quick, A., and Siegle, M. "Distributed Performance Monitoring: Methods, Tools, and Applications". *IEEE Trans. On Parallel and Distributed Systems* 5, 6, 585-598.

[43] Horn, P. "Autonomic Computing: IBM's Perspective on the State of Information Technology,"

http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf (October 2001).

[44] Hsu, W., Smith, A., and Young, H. "Characteristics of Production Database Workloads and the TPC Benchmarks," *IBM Systems Journal* **40**, No. 3 (2001).

[45] Hudson, S. and Smith, I. "Supporting Dynamic Downloadable Appearances in an Extensible User Interface Toolkit". *ACM Proc. of UIST '97,* New York, 159-168, (October 1997).

[46] IBM, *DB2 Universal Database Version 8.1 Administration Guide: Performance*, IBM Corporation, (2003).

[47] IBM, *IBM's Vision of Autonomic Computing: Autonomic Computing Concepts*. *http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf*, (2002).

[48] IBM, *DB2 Intelligent Miner for Data*, http://www-4.ibm.com/software/data/iminer/fordata/about.html, (1999).

[49] IBM, *DB2 Universal Database Version 7 Administration Guide: Performance*, IBM Corporation (2000).

[50] Jain, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, Wiley-Interscience, New York, NY, (April 1991).

[51] Jain, A., Murty, M., and Flynn, P. "Data Clustering: A Review". *ACM Computing Surveys* 31, 3, 264-323, (Sept. 1999).

[52] Keeton, K., and Patterson, D. "Towards a Simplified Database Workload for Computer Architecture Evaluations". *Workload Characterization for Computer System Design*, John, L. and Maynard, A., Eds., Kluwer Academic Publishers, (2000).

[53] Kotsis, G., Krithivasan, K., and Raghavan, S. "A Workload Characterization Methodology for WWW Applications". *Proc. of International Conference on The Performance and Management of Complex Communication Networks (PMCCN'97)*, 145-159.

[54] Kwan, E., Lightstone, S., Storm, A. and Wu, L., IBM Server Group, "Automatic Configuration for IBM DB2 Universal Database". *Online, http://www.redbooks.ibm.com/redpapers/pdfs/redp0441.pdf*.

[55] Landwehr, C. "Formal Models of Computer Security". *ACM Computing Surveys*, 13(3):247–278, (1981).

[56] Letmanyi, H. "Guide on Workload Forecasting". *Special Publication 500-123*, Computer Science and Technology, National Bureau of Standards, Washington, D.C., (March 1985).

[57] Lightstone, S., Lohman, G., and Zilio, D. "Toward Autonomic Computing with DB2 Universal Database", *ACM SIGMOD Record*, (September 2002).

[58] Lo, J., Barroso, L., Eggers, S., Gharachorloo, K., Levy, H., and Parekh, S. "An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors". *Proc. of the 25th Annual International Symposium on Computer Architecture*, 39—50, (June 1998).

[59] Lohman, G., Valentin, G., Zilio, D., Zuliani, M. and, Skelly, A. "DB2 Advisor: An optimizer Smart Enough to Recommend Its Own Indexes", *Proceedings, 16th IEEE Conference on Data Engineering*, San Diego, CA, (2000).

[60] Masters, T. *Neural, Novel & Hybrid Algorithms for Time Series Prediction*, John Wiley & Sons, Inc, New York, NY, (1995).

[61] McCann, J. "The Database Machine: Old Story, New Slant?". *Proceedings of the first Biennial Conference on Innovative Data Systems Research*, VLDB, (January 2003).

[62] Mehrotra, K., Mohan, C., and Ranka, S. *Elements of Artificial Neural Networks,* Cambridge, Massachussetts, MIT Press**,** (1997).

[63] Menascé, D., Almeida , V., and Dowly, L. *Capacity Planning and Performance Modeling: From Mainframes to Client-server Systems*, Prentice Hall, USA, ISBN 0-13-035494-5.

[64] Microsoft. SQL Server 2000 Documentation, Microsoft Corporation, (2002).

[65] Murthy, S. "Automatic Construction of Decision Trees from Data: A Multi-disciplinary Survey," *Data Mining and Knowledge Discovery* **2**, 345—389, (1998).

[66] Office of the Information and Privacy Commissioner, Ontario. *Data Mining: Staking a Claim on Your Privacy*, (January 1998).

[67] Nikolaou, C., Labrinidis, A., Bohn, V., Ferguson, D., Artavanis, M., Kloukinas, C. and Marazakis., M.. *The Impact of Workload Clustering on Transaction Routing*, Technical Report FORTH-ICS TR-238*,* (December 1998).

[68] Oracle. Oracle 9i Manageability Features. An Oracle White Paper, *http://www.oracle.com/ip/deploy/database/oracle9i/collateral/ma_bwp10.pdf*, (September 2001).

[69] Oracle. Oracle 9i Materialized Views. An Oracle White Paper, http://technet.oracle.com/products/oracle9i/pdf/o9i_mv.pdf, (May 2001).

[70] Oracle. *Oracle9iDatabase Performance Guide and Reference*, Release 1(9.0.1), Part# A87503-02, Oracle Corp. (2001).

[71] Oracle. Query Optimization in Oracle 9i. An Oracle White Paper, http://technet.oracle.com/products/bi/pdf/o9i_optimization_twp.pdf, (February 2002).

[72] Patterson, D. "Availability and Maintainability >> Performance: New Focus for a New Century," *USENIX Conference on File and Storage Technologies (FAST '02)*, Keynote Address, Monterey, CA (January 2002).

[73] Patterson D., Brown, A., Broadwell, P., Candea, G., Chen, M., Cutler, J., Enriquez, P., Fox, A., Kiciman, E., Merzbacher, M., Oppenheimer, D., Sastry, N., Tetzlaff, W., Traupman, J., and Treuhaft, N. "Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies", *U.C. Berkeley Computer Science Technical Report, UCB//CSD-02-1175*, University of California, Berkeley (March 15, 2002).

[74] Pentakalos, O. and Menascé, D. "Automated Clustering Based Workload Characterization for Mass Storage Systems". *Fifth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies*, College Park, MD, (September 1996).

[75] Pirahesh, H., Hellerstein, J., and Hasan, W. "Extensible/Rule Based Query Rewrite Optimization in Starburst". *Procs. ACM SIGMOD Conference*, pp. 39-48, (1992).

[76] Pirahesh, H., Leung, T., and W. Hasan, "A Rule Engine for Query Transformation in Starburst and IBM DB2 C/S DBMS", *Procs. IEEE Intl. Conf. On Data Engineering*, pp. 391-400 (1997).

[77] Pitkow, J. and Pirolli, P. Mining Longest Repeating Subsequences to Predict the World Wide Web Surfing. *Proc. of USITS' 99: The 2nd USENIX Symposium on Internet Technologies & Systems*, Boulder, Colorado, USA, (October 1999).

[78] Raghavan, S., Vasukiammaiyar, D. and Haring, G. "Generative Networkload Models for a Single Server Environment". *Proc. ACM SIGMETRICS Conf.*, 118—127.

[79] Rao, J., Zhang, C., Lohman, G., Megiddo, N. , "Automating Physical Database Design in a Parallel Database System", *Proc. 2002 ACM SIGMOD*, Madison, WI, (2002).

[80] Rohlf, F. "Algorithm 76: Hierarchical Clustering Using the Minimal Spanning Tree". In *The Computer Journal* 16, 93-95, (1973).

[81] Sapia, C. "PROMISE: Predicting Query Behavior to Enable Predictive Caching Strategies for OLAP Systems,". *Proc. of the Second International Conference on Data Warehousing and Knowledge Discovery (DAWAK 2000)*, 224-233, (2000).

[82] Schiefer, B. and Valentin, G. "DB2 Universal Database Performance Tuning", *IEEE Data Engineering Bulletin*, 22(2), pp. 12-19, (June 1999).

[83] Schiff, D. and D'Agostino, R. *Practical Engineering Statistics*, Wiley-Interscience, ISBN 0471547689.

[84] Shafer, J., Agrawal, R., Mehta, M. "SPRINT: A Scalable Parallel Classifier for Data Mining," *Proc. of the 22th Int'l Conference on Very Large Databases*, Mumbai (Bombay), India, (September 1996).

[85] Stillger, M., Lohman, G., Markl, V., and Kandil, M.. "LEO - DB2's LEarning Optimizer", *VLDB 2001*, Rome, Italy, pp. 19-28, (2001).

[86] TPC. *TPC Benchmark D Standard Specification Revision 2.1*, Transaction Processing Performance Council (1999).

[87] TPC. *TPC Benchmark H Standard Specification Revision 1.3.0*, Transaction Processing Performance Council (1999).

[88] TPC. *TPC Benchmark W (Web Commerce) Standard Specification Revision 1.7*, Transaction Processing Performance Council (October 2001).

[89] TPC. *TPC Benchmark C Standard Specification Revision 5.0*, Transaction Processing Performance Council (February 2001).

[90] TPC. *TPC Benchmark H Standard Specification Revision 1.3.0*, Transaction Processing Performance Council (1999).

[91] TPC. *TPC Benchmark W (Web Commerce) Standard Specification Revision 1.7*, Transaction Processing Performance Council (October 2001).

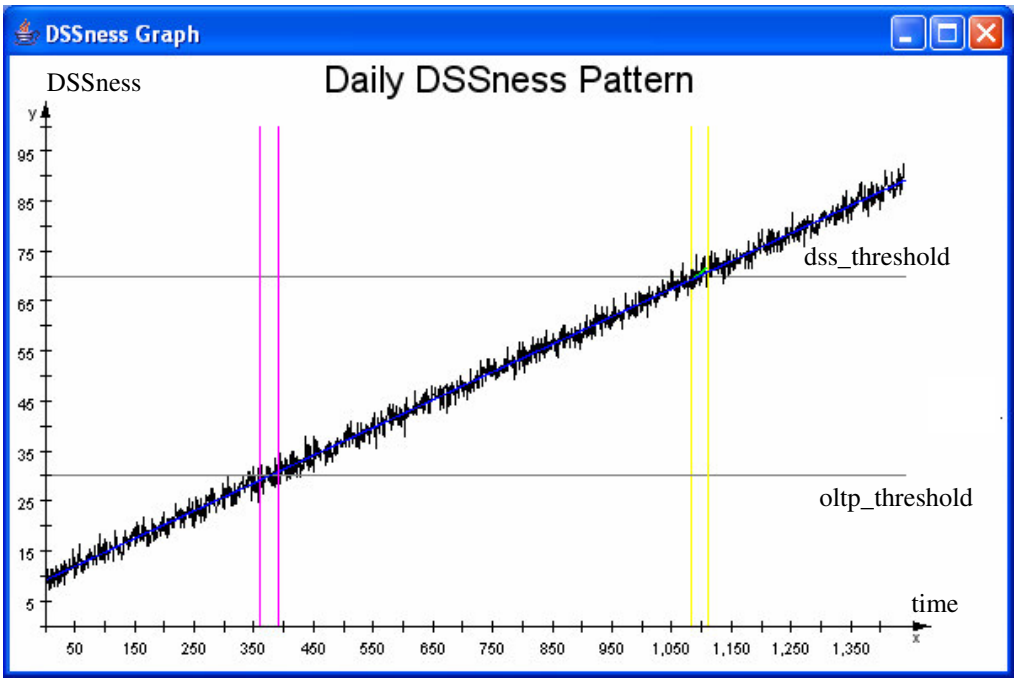
[92] Turner, T. *A Beginner's Guide To Day Trading Online*. Adams Media Corp., first edition, (2000).

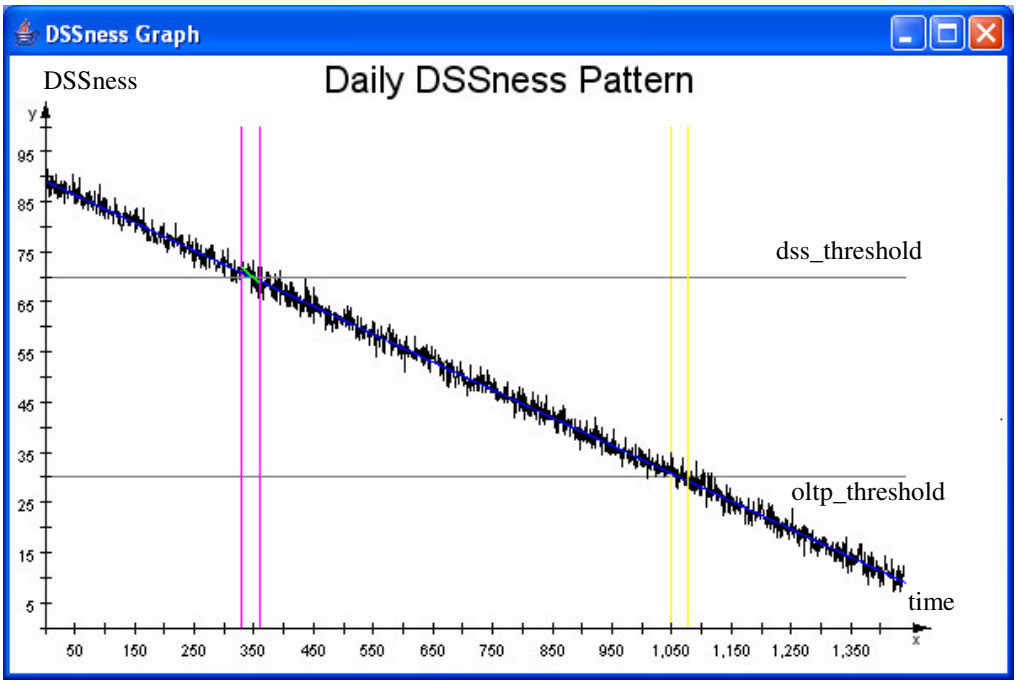
[93] Weikum, G., Mönkeberg, A., Hasse, C., and Zabback, P. "Self-tuning Database Technology and Information Services: From Wishful Thinking to Viable Engineering". *VLDB 2002*, pp. 20-31, (2002).

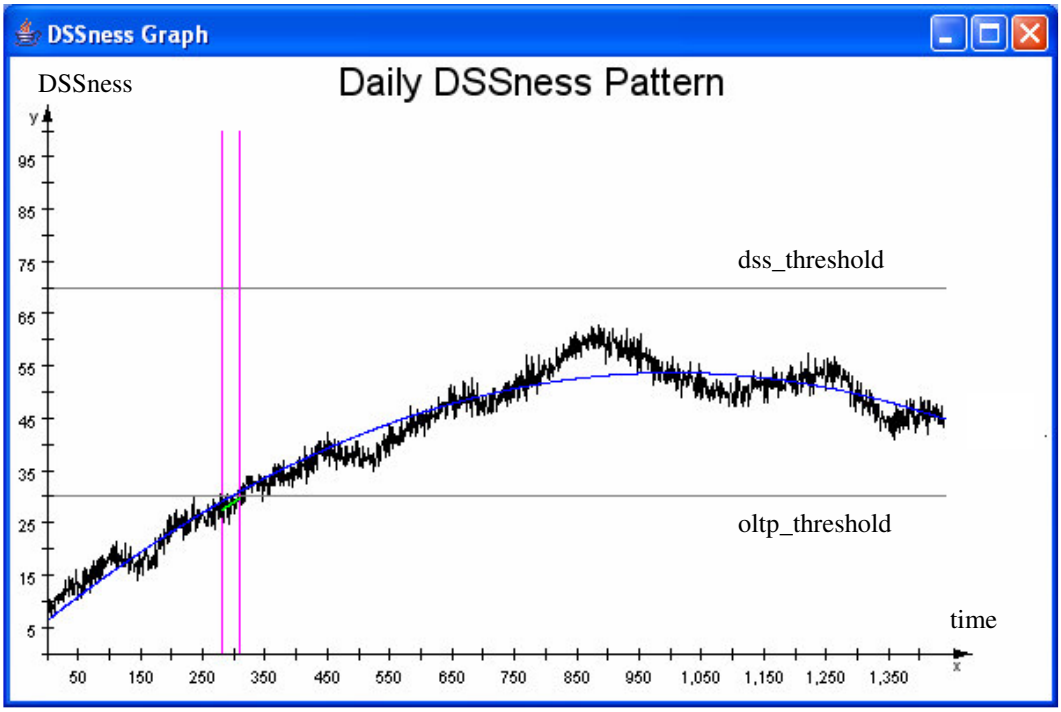
[94] Yu, P., and Dan, A. "Performance Analysis of Affinity Clustering on Transaction Processing Coupling Architecture," *IEEE Transactions on Knowledge and Data Engineering* 6, **5**, 764-786, (October 1994).

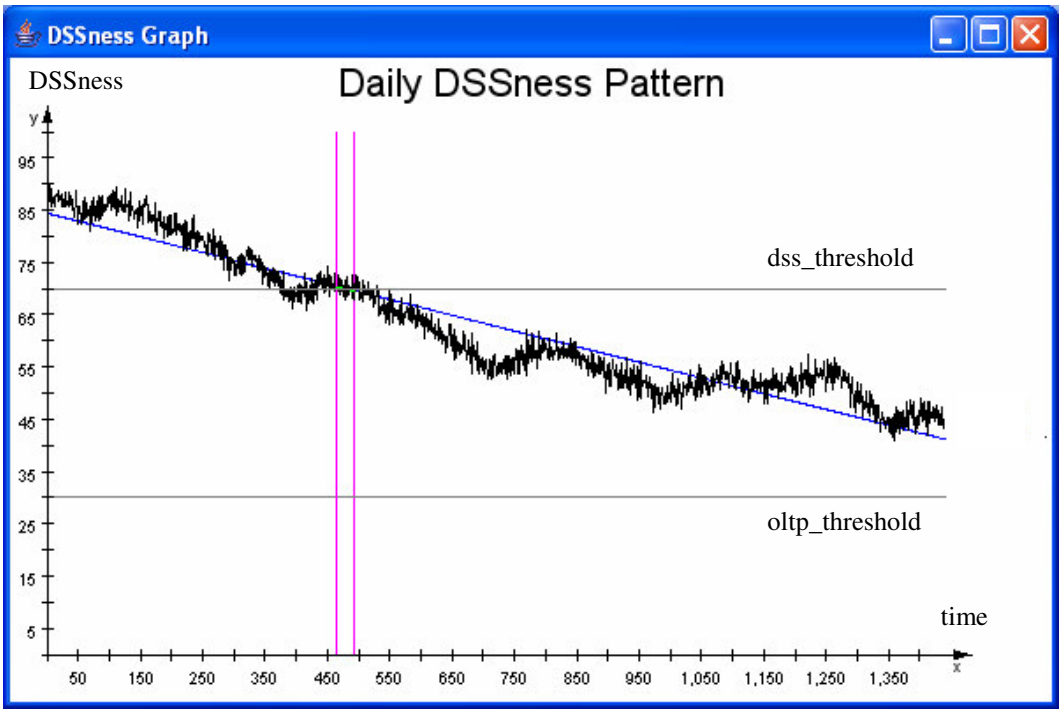
[95] Zaïane, O., Xin, M. and Han, J. "Discovering Web Access Patterns and Trends by Applying OLAP and Data Mining Technology on Web Logs". *Proc. Advances in Digital Libraries Conf. (ADL'98)*, Santa Barbara, CA, 19-29, (April 1998).

APPENDIX A: EXAMPLES OF DSSNESS SCENARIOS






DSSness Graph
DSSness
Daily DSSness Pattern
y
95
85
75
65
55
45
35
25
15
5
dss_threshold
oltp_threshold
time
50
150
250
350
450
550
650
750
850
950
1,050
1,150
1,250
1,350
x


DSSness Graph
DSSness
Daily DSSness Pattern
y
95
85
75
65
55
45
35
25
15
5
dss_threshold
oltp_threshold
time
50
150
250
350
450
550
650
750
850
950
1,050
1,150
1,250
1,350
x

APPENDIX B: GLOSSARY OF TERMS

ADBMS (Autonomic Database Management System) – a DBMS that exhibits the characteristics of an autonomic system by being self-configuring, self-optimizing, self-healing, and self-protecting.

AC (Autonomic Computing) - an initiative started by IBM in 2001. Its ultimate aim is to create self-managing computer systems to overcome their rapidly growing complexity and to enable their further growth.

Browsing Profile – it is a TPC-W workload profile characterized by extensive browsing and searching activities.

Classifier(C, H) - workload classifier built by training it on samples from the TPC-C and TPC-H workloads.

Classifier(O, B) - workload classifier built by training it on samples from the Ordering and Browsing profiles of TPC-W.

DBA (Database Administrator) - a person who is responsible for the environmental aspects of a database. In general, these include: Recoverability, Integrity, Security, Availability, and Performance.

DBMS (Database Management System) - a computer program (or more typically, a suite of them) designed to manage a database, a large set of structured data, and run operations on the data requested by numerous users.

DSS (Decision Support System) Workload - workload consisting of decision-support queries of high complexity and low volume.

DSSness – a percentage that determines the concentration of the DSS type vs. OLTP type in a workload sample.

Dynamic Characteristics – the properties that describe the workload behavior over time (e.g., how the workload type changes during the day).

HC (Hybrid Classifier) – it is a workload trained on different flavors of OLTP workloads, namely TPC-C and the Ordering Profile, and on different flavors of DSS workloads, namely TPC-H and the Browsing Profile.

GHC (Graduated Hybrid Classifier – it is an HC (Hybrid Classifier) but has the ability to recognize different shades of DSS workloads and OLTP workloads.

MUM (Model Update Mechanism) – a component in the in the Psychic-Skeptic architecture that performs regular workload sampling to keep prediction models up to date.

OLTP (On-Line Transaction Processing) - an OLTP workload consists of high volume of transactions, and a few simple queries.

Ordering Profile – it is a TPC-W workload profile characterized by extensive ordering activities.

Performance Snapshot – a set of low-level performance measures (or attributes) collected at some point in time as the DBMS processes some workload. Snapshots are the basic objects for the classification algorithm used to build the Workload Classifier.

Psychic – it is a component in the Psychic-Skeptic architecture that performs off-line prediction for workload type shifts by analyzing historical data.

Shopping Profile - it is a TPC-W workload profile that exhibits some product ordering activities but browsing is still dominant.

Skeptic – it is a component in the Psychic-Skeptic architecture that works on-line by sampling the workload for small intervals in order to validate every shift predicted by the Psychic.

Static Characteristics - workload properties that provide a general description of the workload with no respect to how these properties may change over time (e.g., the dominant workload type, or the average requests submitted to the system per day).

TCO (Total Cost of Ownership) - a type of calculation designed to help consumers and enterprise managers assess direct and indirect costs as well as benefits related to the purchase of computer software or hardware.

TPC (Transaction Processing Performance Council) – an organization that produces industry-standard benchmarks for DBMSs.

TPC-C Workload – an example of a hardcore OLTP workload. TPC-C simulates a complete environment where a population of terminal operators executes transactions against a database. The benchmark is centered around the principal activities (transactions) of an order-entry environment.

TPC-H Workload – an example of a hardcore DSS workload. TPC-H illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions.

TPC-W Workload - comprises a set of basic operations designed to exercise transactional web system functionality in a manner representative of internet commerce application environments. User activities are described by three profiles: Browsing, Shopping, and Ordering.

TrainingDataModel - is a component in the Psychic-Skeptic architecture that stores historical data of DSSness of a number of days. TrainingDataModel provides a

number of analytical functions that operate on these data to other components in the architecture.

Workload - a set of requests, or components, that place different demands on various system resources.

Workload Characterization – a process by which a representative, compact, and accurate model of a system's workload is built. The model should be able to describe and reproduce the dynamic behavior of the workload and its most essential static features.

Workload Classifier – a tool that can assess the relative concentration (percentage) of each workload type in a given workload sample.

Workload Predictor – a tool that can predict major shifts in the workload type.