# DBSeer: Pain-free Database Administration through Workload Intelligence*

Dong Young Yoon
University of Michigan
Ann Arbor
dyoon@umich.edu

Barzan Mozafari
University of Michigan
Ann Arbor
mozafari@umich.edu

Douglas P. Brown
Teradata Inc.
San Diego
doug.brown@teradata.com

## ABSTRACT

The pressing need for achieving and maintaining high performance in database systems has made database administration one of the most stressful jobs in information technology. On the other hand, the increasing complexity of database systems has made qualified database administrators (DBAs) a scarce resource. DBAs are now responsible for an array of demanding tasks; they need to (i) provision and tune their database according to their application requirements, (ii) constantly monitor their database for any performance failures or slowdowns, (iii) diagnose the root cause of the performance problem in an accurate and timely fashion, and (iv) take prompt actions that can restore acceptable database performance.

However, much of the research in the past years has focused on improving the raw performance of the database systems, rather than improving their manageability. Besides sophisticated consoles for monitoring performance and a few auto-tuning wizards, DBAs are not provided with any help other than their own many years of experience. Typically, their only resort is trial-and-error, which is a tedious, ad-hoc and often sub-optimal solution.

In this demonstration, we present DBSeer, a workload intelligence framework that exploits advanced machine learning and causality techniques to aid DBAs in their various responsibilities. DBSeer analyzes large volumes of statistics and telemetry data collected from various log files to provide the DBA with a suite of rich functionalities including performance prediction, performance diagnosis, bottleneck explanation, workload insight, optimal admission control, and what-if analysis. In this demo, we showcase various features of DBSeer by predicting and analyzing the performance of a live database system. Will also reproduce a number of realistic performance problems in the system, and allow the audience to use DBSeer to quickly diagnose and resolve their root cause.

## 1. INTRODUCTION

Many enterprise applications rely on database management systems to store and query their business data. This increasing reliance on database systems has made performance requirements less forgiving; any performance degradation in such databases can directly breed customer discontent and cause revenue loss. Thus, a major responsibility of DBAs is to sustain database performance 24/7, and at a level that can meet the mission-critical requirements of the underlying business operations. To fulfill this goal, DBAs are required to perform various operations:

- **Database Provisioning.** DBAs need to provision their database with the resources necessary to meet the application level SLAs (service-level agreements). For example, DBAs need to determine the best hardware configuration given a fixed budget or a target peak load, e.g., how to best allocate N dollars between more/faster processors, more memory, larger SSDs, and faster disks.

- **Database Tuning.** DBAs need to design and tune their database for their target workload. They also need to define user quotas and admission control policies (a.k.a. throttles) to minimize SLA violations.

- **Performance Diagnosis.** A major responsibility of DBAs in large organizations is to constantly monitor their database for any performance failures or slowdowns. However, diagnosing the root cause of performance problems is a tedious task, as it requires the DBA to consider many possibilities by manually inspecting various log files and queries executed over time. These challenges are exacerbated in highly concurrent workloads (e.g., shared databases in large organizations and OLTP workloads), where performance problems cannot be traced back to a few demanding queries or their poor execution plans. Often, each query takes only a small fraction of the overall resources to complete. However, tens or hundreds of concurrent queries competing for the same resources (e.g., CPU, network, disk I/O, memory) can create highly non-linear and counter-intuitive effects on database performance.

- **Performance Mending.** Once the root cause of a performance problem is found, appropriate actions need to be taken to restore acceptable database performance. Though the necessary action may be straightforward in some cases, in many cases several alternatives may be present. Besides their past experience, most DBAs simply rely on trial and error to find the most appropriate action.

*DBSeer is an open-source tool and can be downloaded at http://dbseer.org and a video demonstration of its features can be found at http://dbseer.org/video


*Proceedings of the VLDB Endowment,* Vol. 8, No. 12

Given the complexity of today's database management systems, these responsibilities are quite daunting for all but the most experienced of DBAs. As a result of this increasing demand, qualified database administrators (DBAs) have become a scarce resource [1].

Over the past few years, we have developed an open-source framework, called DBSeer, that integrates several functionalities to assist DBAs in the tasks above [6, 5]. DBSeer is not meant to replace an experienced DBA. Rather, DBSeer is designed to alleviate DBAs from their current trial-and-error procedures—which are tedious, adhoc and error-prone—by offering them a set of automated, principled and more accurate alternatives.

To enable accurate provisioning, DBSeer uses machine learning and statistical regression techniques to identify the bottleneck resources and predict performance for a given set of resources. These features help DBAs decide how to best allocate their budget to various types of resources. For instance, by analyzing the current workload, DBSeer might predict that memory will be the bottleneck resource if the load on the system were to triple [6]. DBSeer also summarizes and mines past workloads, providing DBAs further insight into their own workload and underlying applications.[1]

DBSeer also comes with a performance explanation module, called DBSherlock. DBSherlock utilizes the statistics collected from the database and the operating system. By combining techniques from outlier detection and causality analysis, DBSherlock assist DBAs in diagnosing performance problems more easily, more accurately, and in a principled manner. Through a graphical user interface, the DBA can specify certain instances of past performance that he/she deems abnormal; the DBSherlock module then automatically analyzes past statistics and many possible explanations to suggest the most likely cause of the user-perceived performance anomaly.

Finally, DBSeer uses the same performance prediction features described above to enable *what-if* analysis [5]. This features allows the DBA to predict the outcome of certain actions on performance before taking those actions. For example, in an overloaded system, the DBA can ask whether halving the number of queries issued by a certain user will bring the CPU usage back to 80%.

We propose to demonstrate DBSeer in action. We believe that a system designed for making database administration easier is best appreciated when tested and used in person. To provide this first-hand experience, we will re-produce a set of realistic performance problems in a live database, and then ask our audience to use DBSeer's various features to diagnose and rectify the performance problem himself/herself. The audience will observe how DBSeer can predict performance, tune the database, or provide performance diagnosis and automatic explanations. DBSeer is currently available as an open-source, third-party tool.[2]

The rest of this paper is organized as follows. Section 2 provides a high-level overview of DBSeer's architecture and it various modules. Section 3 describes our demonstration scenario at VLDB. We conclude in Section 4.

# 2. DBSEER ARCHITECTURE

DBSeer's architecture consists of several major components, as depicted in Figure 1. Next, we describe each of these components.

## 2.1 Data Collection

DBSeer collects various aggregate statistics from the database (e.g., MySQL, Postgres, or MariaDB) and the operating system

[1]To enable better database, we have developed a different tool, called CliffGuard [7]. Unlike existing tuning advisors, CliffGuard finds a robust physical design that is resilient against sudden changes of the workload; see http://cliffguard.org.

[2]http://dbseer.org

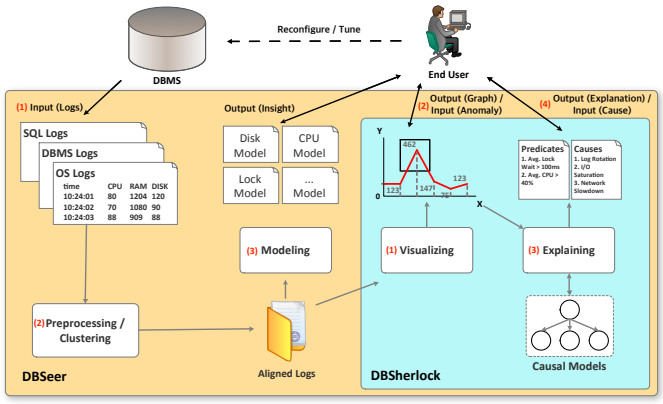


Figure 1: DBSeer Architecture

(Linux in our case) *in situ*, via their standard logging features. This is shown as component (1) in Figure 1. These statistics are collected at one-second intervals and consist of the following:

1. Aggregated OS statistics, including per-core CPU usage, number of disk I/Os, number of network packets, number of page faults, number of context switches.
2. Workload statistics from the DBMS, including the number of SELECT, UPDATE, DELETE and INSERT commands executed, number of flushed and dirty pages, and the total lock wait-time.
3. Timestamped query logs, containing start-time, duration, and the SQL statements executed by the system.

## 2.2 Automatic Workload Clustering/Classification

DBSeer further processes the collected data for performance modeling, prediction, and diagnosis (component (2) in Figure 1). First, DBSeer uses the collected statistics to cluster the transactions and queries into a set of types or classes that exhibit a similar access pattern, i.e., they perform similar operations on each table or exhibit similar patterns of resource usage.

Extracting query types allows us to model and categorize different queries in the workload. DBSeer parses the query logs and extracts features, called *query summaries*, including the tables accessed, locking mode (e.g., exclusive or shared), the approximate number of row accessed, the time between statements. DBSeer uses these extracted features to apply the DBSCAN [3] clustering algorithm, which groups individual queries into a number of categories based on their access patterns.

DBSeer also computes aggregate performance statistics for each time interval (e.g., the average and quantile latencies, total page-faults, etc.). These aggregate statistics are then aligned with the OS and DBMS statistics according to their timestamps. The aligned, timestamped logs are processed by DBSeer for performance modeling, prediction and explanation, as described next.

## 2.3 Performance Prediction

DBSeer utilizes both black-box and white-box models for predicting various types of performance metrics. The black-box models work quite well for certain resources such as CPU, network, and log writes. These models make minimal assumptions about the underlying system, and hence are not specific to a particular DBMS. White-box models are needed for other resources (e.g., locking, page flushes due to log recycling) when making predictions about a drastically different workload than the one observed during training. Our white-box models include an iterative model for log recycling, as well as a number of optimizations to Thomasians's model of two-phase locking (2PL) [8].

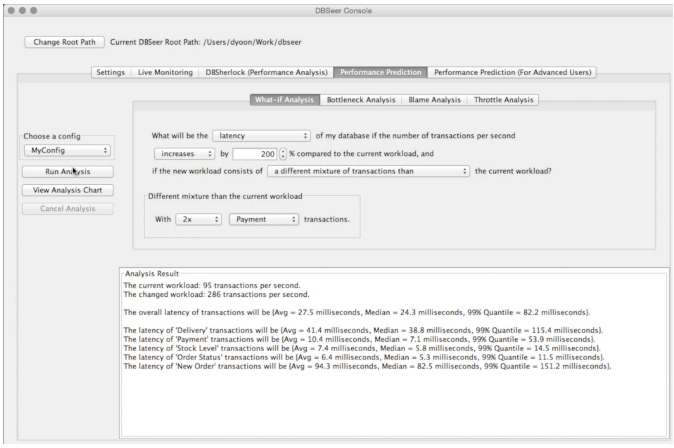


Figure 2: DBSeer's performance prediction console

Using these statistical regression models, DBSeer provides several features (see Figure 2):

1. **What-if Analysis.** Users can inquire about unseen situations, e.g., what will happen to 99% latencies if the overall load triples? Or what will happen to the CPU usage if the total number of 'payment' transactions doubles? This feature enables DBAs to better provision and tune their database to meet their performance goals. This is far more accurate and less tedious than the current trial-and-error approach.

2. **Bottleneck Analysis.** Users can pose questions about the maximum throughput that their database can sustain. They can also ask which resource (e.g., CPU, disk, or locks) will bottleneck first if their load increases. This feature provides great insight into the workload characteristics, e.g., buying faster disks may not help with a workload that is lock-bound.

3. **Blame Analysis.** When the system is overload, users can ask which *type* of transactions is most responsible for the high CPU usage, disk usage, or lock contention in the system. For example, in the TPC-C benchmark, the 'order status' and 'payment' transactions are most repressible for high disk reads and writes, respectively. This features helps DBAs decide which parts of their workload should be further inspected, optimized, or migrated to a separate server.

4. **Throttle Analysis (Optimal Admission Control).** When the overall load exceeds available resources, performance metrics plummet and SLAs are violated. In these situation, a critical question is which queries or transactions need to be throttled in order to minimize the SLA violations. For instance, given a different penalty for throttling each type of transaction, DBSeer can determine the maximum number of each type of transaction to guarantee a certain 99% latency.

## 2.4 DBSherlock

DBSherlock is an integrated module in DBSeer, to perform performance diagnosis and explanations. DBSherlockutilizes the same input data gathered from DBSeer's data collection and preprocessing (components (1), (2) in Figure 1). The DBSherlock module performs performance explanation and diagnosis in four steps, as shown in Figure 1.

1. **Visualization.** Through our graphical user interface, the end user (e.g., a DBA) can generate scatter plots of various performance statistics of the DBMS over time.

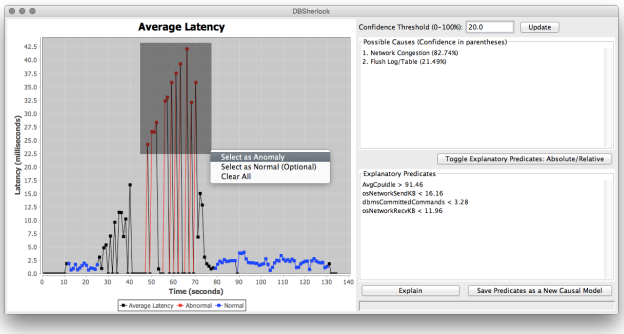


Figure 3: Screenshot of DBSherlock's user interface

2. **Anomaly Detection.** If the end user deems any of the performance metrics of the DBMS unexpected, abnormal, or suspicious in any period of time, s/he can select that region of the plot and query DBSherlock for an explanation of the observed anomaly or performance problem.

3. **Anomaly Explanation.** Given the user-specified region of anomaly, DBSherlock analyzes the collected statistics and explains the anomaly using either descriptive predicates or actual causes.

4. **Anomaly Diagnosis and User Feedback.** Once a DBA has diagnosed the actual cause of the observed performance problem by looking at DBSherlock's explanations, s/he provides evaluative feedback to DBSherlock. This feedback is then incorporated in DBSherlock as a causal model and used for improving future explanations.

Figure 3 is a screenshot of DBSherlock's graphical user interface, where users can plot a graph of various performance metrics over their time window of interest. For instance, users may plot the average latency of queries, number of disk I/Os, or CPU usage over the past hour, day or week. Here, Figure 3 shows a scatter plot of the average latency of queries over time. The user can select some region(s) of the graph where she finds some metrics abnormal, suspicious, counter-intuitive, or simply worthy of an explanation. Regardless of the user's particular reason, we simply call the selected region(s) an *anomaly* (or *abnormal* regions). Optionally, the user can also specify other regions of the graph where s/he thinks are normal (otherwise, the rest of the graph is implicitly considered as normal). After specifying the regions, users can query DBSherlock to find likely causes or explanations for the observed *anomaly*.

Given a user-specified anomaly, DBSherlock provides explanations in one of the following forms:

(i) predicates over different attributes of the input data; or

(ii) likely causes (and their corresponding confidence) based on existing causal models.

DBSherlock generates a number of predicates that best explain the anomaly by identifying anomalous values of some of the attributes in the input data. The generated predicates are conjunctive. For example, DBSherlock may explain an anomaly caused by a network slowdown by generating the following predicates:

$$\texttt{network\_send} < 10\text{KB} \wedge \texttt{network\_recv} < 10\text{KB} \wedge \texttt{client\_wait\_times} > 100\text{ms} \wedge \texttt{cpu\_usage} < 5$$

showing that there are clients waiting without much CPU activity. These predicates are generated by our novel predicate generation algorithm, which aims to find predicates that segregate the input tuples in the user-perceived abnormal region well from those

| Type of anomaly | Description |
|---|---|
| Workload Spike | Greatly increase the rate of transactions and the number of clients simulated by OLTPBenchmark (128 additional terminals with transaction rate of 50,000). |
| I/O Saturation | Invoke stress-ng, which spawns multiple processes that spin on write()/unlink()/sync() system calls. |
| Table Restore | Dump the pre-dumped *history* table back into the database instance. |
| CPU Saturation | Invoke stress-ng, which spawns multiple processes calling poll() system calls to stress CPU resources. |
| Flush Log/Table | Flush all tables and logs by invoking *mysqladmin* commands ('flush-logs' and 'refresh'). |
| Network Congestion | Simulate network congestion by adding an artificial 300-milliseconds delay to every traffic over the network via Linux's *tc* (Traffic Control) command. |
| Lock Contention | Change the transaction mix to execute *NewOrder* transactions only on a single warehouse and district. |
| Poorly Written Query | Execute a poorly written JOIN query, which would run efficiently if written properly. |
| Poor Physical Design | Create unnecessary indexes on tables where INSERT statements are mostly executed. |

Table 1: Nine types of performance anomalies

in the normal regions. Once the user identifies the actual problem (network congestion, in this example) using these predicates as diagnostic hints, she can provide feedback to DBSherlock by accepting these predicates and labeling them with the actual cause found. This 'cause' and its corresponding predicates comprise a causal model, which then is utilized by DBSherlock for future diagnoses.

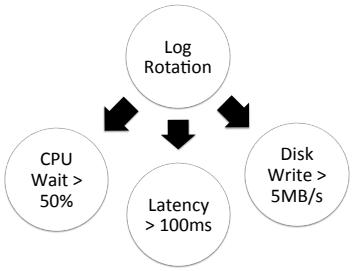


Figure 4: An example of a causal model in DBSherlock

DBSherlock uses a simplified version of the *causal model* proposed in the seminal work of Halpern and Pearl [4]. Our causal model couples the user-labeled, actual 'cause' with its corresponding predicates from the user's feedback. Figure 4 is an example of a causal model with 'Log Rotation' as the cause and three corresponding predicates. When there are any causal models in the system of DBSherlock from previous diagnoses, DBSherlock calculates the *confidence* of every existing causal model for the given anomaly. This *confidence* measures a causal model's fitness for the given situation. The use of the causal model enables DBSherlock to generate explanations that are more descriptive and informative than just predicates, facilitating a fast and accurate diagnosis of the performance problem.

# 3. DEMONSTRATION DETAILS

We will collect live data by running various types of workloads using MySQL and OLTPBenchmark [2]. We will show DBSeer's visualization and live monitoring features to our audience using the collected data. Then, we will demonstrate DBSeer's capability in performance and resource prediction. The audience can then use DBSeer to predict various performance metrics and perform various type of analysis, e.g., what-if analysis, blame analysis, throttle analysis, and so on. The users will compare DBSeer's predictions against the actual system.

We will also have input logs collected from a number of scenarios for reproducing different types of performance anomalies. These performance anomalies will include some of the most common problems faced in every-day operations of any DBMS, such as network congestion, I/O saturation, database maintenance, as listed in Table 1. We will demonstrate how users can interact with the DBSherlock module in DBSeervia its graphical user interface. Our audience will inspect some of the graphs on aggregated statistics and ask DBSherlock for explanations by selecting the abnormal region(s). Before revealing the true cause, we will first ask the audience to provide an explanation for each anomaly, based on their own intuition. We will then allow them to invoke DBSherlock to display the most likely causes as an explanation for the user-perceived anomaly. By comparing their accuracy before and after seeing DBSherlock's explanations, the audience will be able to judge DBSherlock's effectiveness in finding the actual cause of performance problems.

# 4. CONCLUSION

DBSeer performs statistical performance modeling and prediction, assisting a DBA in understanding how database resource consumption and performance vary as load on the system changes. DBSeer can also explain performance anomalies in the context of a complex database system. Currently, such tasks are performed manually by highly-skilled and highly-paid DBAs, as they spend many hours inspecting various log files and queries. Consequently, it is a tremendous cost to the enterprise when DBAs, who are scarce resources themselves, spend much of their time on these tedious tasks. In this regard, DBSeer helps DBAs maintain their database performance with much less effort. Our demonstration will highlight the main functionalities of DBSeer and its accuracy in performance prediction and explanation, focusing on the usability of our framework by a general audience.

# 5. REFERENCES

[1] The most wanted jobs in IT. http://tinyurl.com/odezqov, 2014.

[2] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudre-Mauroux. Oltp-bench: An extensible testbed for benchmarking relational databases. *PVLDB*, 7, 2013.

[3] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 1996.

[4] J. Y. Halpern and J. Pearl. Causes and explanations: a structural-model approach. part i: causes. In *UAI*, 2001.

[5] B. Mozafari, C. Curino, A. Jindal, and S. Madden. Performance and resource modeling in highly-concurrent OLTP workloads. In *SIGMOD*, 2013.

[6] B. Mozafari, C. Curino, and S. Madden. Dbseer: Resource and performance prediction for building a next generation database cloud. In *CIDR*, 2013.

[7] B. Mozafari, E. Z. Y. Goh, and D. Y. Yoon. Cliffguard: A principled framework for finding robust database designs. In *SIGMOD*, 2015.

[8] A. Thomasian. On a more realistic lock contention model and its analysis. In *ICDE*, 1994.