DIADS: Addressing the "My-Problem-or-Yours" Syndrome with Integrated SAN and Database Diagnosis

Shivnath Babu
Duke University
shivnath@cs.duke.edu

Nedyalko Borisov
Duke University
nedyalko@cs.duke.edu

Sandeep Uttamchandani
IBM Almaden Research Center
sandeepu@us.ibm.com

Ramani Routray
IBM Almaden Research Center
routrayr@us.ibm.com

Aameek Singh
IBM Almaden Research Center
singh@us.ibm.com

Abstract

We present DIADS, an integrated *DIA*gnosis tool for *D*atabases and *S*torage area networks (SANs). Existing diagnosis tools in this domain have a database-only (e.g., [11]) or SAN-only (e.g., [28]) focus. DIADS is a first-of-a-kind framework based on a careful integration of information from the database and SAN subsystems; and is not a simple concatenation of database-only and SAN-only modules. This approach not only increases the accuracy of diagnosis, but also leads to significant improvements in efficiency.

DIADS uses a novel combination of non-intrusive machine learning techniques (e.g., Kernel Density Estimation) and domain knowledge encoded in a new symptoms database design. The machine learning component provides core techniques for problem diagnosis from monitoring data, and domain knowledge acts as checks-and-balances to guide the diagnosis in the right direction. This unique system design enables DIADS to function effectively even in the presence of multiple concurrent problems as well as noisy data prevalent in production environments. We demonstrate the efficacy of our approach through a detailed experimental evaluation of DIADS implemented on a real data center testbed with PostgreSQL databases and an enterprise SAN.

1 Introduction

"The online transaction processing database myOLTP has a 30% slow down in processing time, compared to performance two weeks back." This is a typical problem ticket a database administrator would create for the SAN administrator to analyze and fix. Unless there is an obvious failure or degradation in the storage hardware or the connectivity fabric, the response to this problem ticket would be: *"The I/O rate for myOLTP tablespace volumes has increased 40%, with increased sequential reads, but the response time is within normal bounds."* This to-and-fro may continue for a few weeks, often driving SAN administrators to take drastic steps such as migrating the database volumes to a new isolated storage controller or creating a dedicated SAN silo (the inverse of *consolidation*, explaining in part why large enterprises still continue to have highly under-utilized storage systems). The *myOLTP* problem may be fixed eventually by the database administrator realizing that a change in a table's properties had made the plan with sequential data scans inefficient; and the I/O path was never an issue.

The above example is a realistic scenario from large enterprises with separate teams of database and SAN administrators, where each team uses tools specific to its own subsystem. With the growing popularity of Software-as-a-Service, this division is even more predominant with application administrators belonging to the customer, while the computing infrastructure is provided and maintained by the service provider administrators. The result is a lack of end-to-end correlated information across the system stack that makes problem diagnosis hard. Problem resolution in such cases may require either *throwing iron* at the problem and re-creating resource silos, or employing highly-paid consultants who understand both databases and SANs to solve the performance problem tickets.

The goal of this paper is to develop an integrated diagnosis tool (called DIADS) that spans the database and the underlying SAN consisting of end-to-end I/O paths with servers, interconnecting network switches and fabric, and storage controllers. The input to DIADS is a problem ticket from the administrator with respect to a degradation in database query performance. The output is a collection of top-K events from the database and SAN that are candidate root causes for the performance degradation. Internally, DIADS analyzes thousands of entries in the performance and event logs of the database and individual SAN devices to shortlist an extremely selective subset for further analysis.

1.1 Challenges in Integrated Diagnosis

Figure 1 shows an integrated database and SAN taxonomy with various logical (e.g., sort and scan operators in a database query plan) and physical components (e.g., server, switch, and storage controller). Diagnosis of problems within the database or SAN subsystem is an

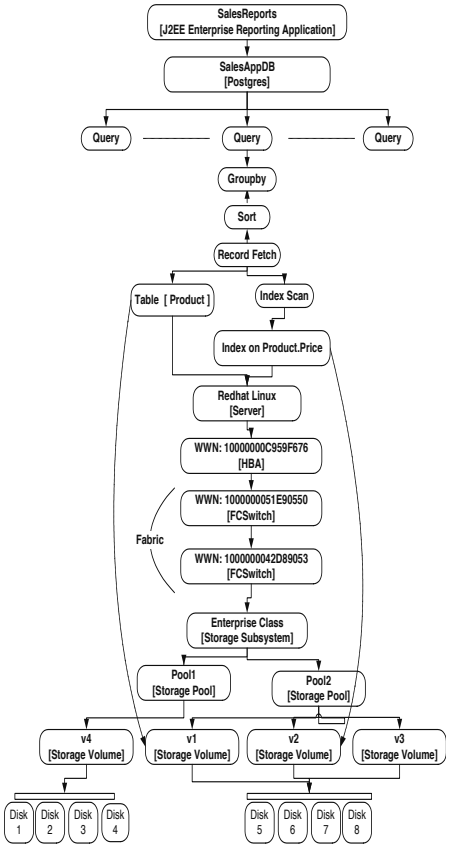

Figure 1: Example database/SAN deployment.

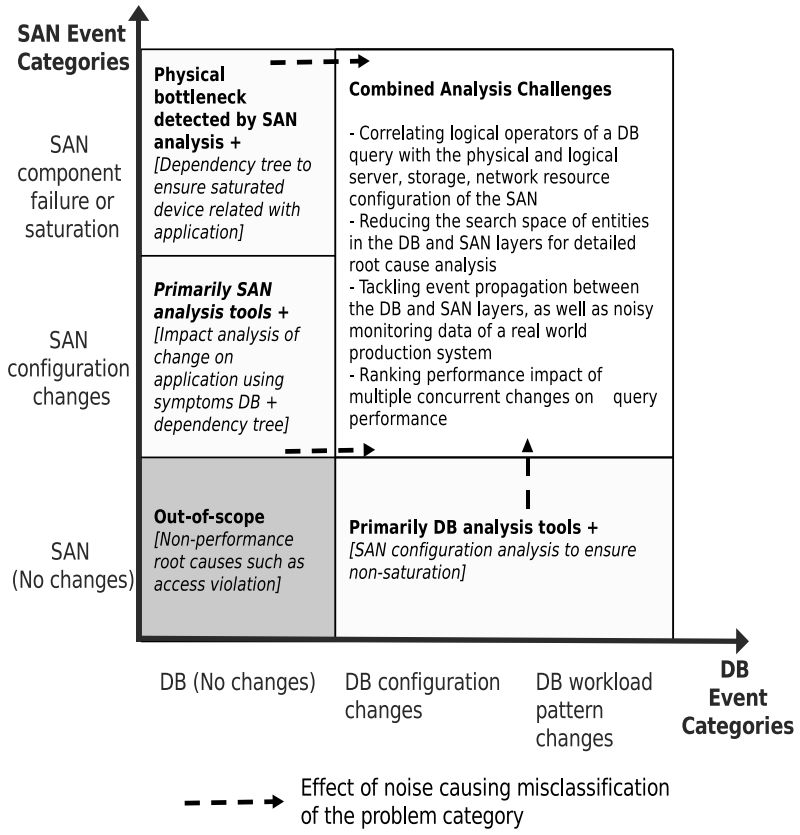

Figure 2: Taxonomy of scenarios for root-cause analysis.

area of ongoing research (described later in Section 2). Integrated diagnosis across multiple subsystems is even more challenging:

- *High-dimensional search space*: Integrated analysis involves a large number of entities and their combinations (see Figure 1). Pure machine learning techniques that aim to find correlations in the raw monitoring data—which may be effective within a single subsystem with few parameters—can be ineffective in the integrated scenario. Additionally, real-world monitoring data has inaccuracies (i.e., the data is *noisy*). The typical source of noise is the large monitoring interval (5 minutes or higher in production environments) which averages out the instantaneous effects of spikes and other bursty behavior.
- *Event cascading and impact analysis*: The cause and effect of a problem may not be contained within a single subsystem (i.e., *event flooding* may result). Analyzing the impact of an event across multiple subsystems is a nontrivial problem.
- *Deficiencies of rule-based approaches*: Existing diagnosis tools for some commercial databases [11] use a rule-based approach where a root-cause taxonomy is created and then complemented with rules to map observed symptoms to possible root causes. While this approach has the merit of encoding valuable domain knowledge for diagnosis purposes, it may become complex to maintain and customize.

1.2 Contributions

The taxonomy of problem determination scenarios handled by DIADS is shown in Figure 2. The events in the SAN subsystem can be broadly classified into configuration changes (such as allocation of new applications, change in interconnectivity, firmware upgrades, etc.) and component failure or saturation events. Similarly, database events could correspond to changes in the configuration parameters of the database, or a change in the workload characteristics driven by changes in query plans, data properties, etc. The figure represents a matrix of change events, with relatively complex scenarios arising due to combinations of SAN and database events. In real-world systems, the *no change* category is misleading, since there will always be change events recorded in management logs that may not be relevant or may not impact the problem at hand; those events still need to be filtered by the problem determination tool. For completeness, there is another dimension (outside the scope of this paper) representing transient effects, e.g., workload con-

tention causing transient saturation of components.

The key contributions of this paper are:

- A novel workflow for integrated diagnosis that uses an end-to-end canonical representation of database query operations combined with physical and logical entities from the SAN subsystem (referred to as *dependency paths*). DIADS generates these paths by analyzing system configuration data, performance metrics, as well as event data generated by the system or by user-defined triggers.
- The workflow is based on an innovative combination of machine learning, domain knowledge of configuration and events, and impact analysis on query performance. This design enables DIADS to address the integrated diagnosis challenges of high-dimensional space, event propagation, multiple concurrent problems, and noisy data.
- An empirical evaluation of DIADS on a real-world testbed with a PostgreSQL database running on an enterprise-class storage controller. We describe problem injection scenarios including combinations of events in the database and SAN layers, along with a drill-down into intermediate results given by DIADS.

2 Related Work

We give an overview of relevant database (DB), storage, and systems diagnosis work, some of which is complementary and leveraged by our integrated approach.

2.1 Independent DB and Storage Diagnosis

There has been significant prior research in performance diagnosis and problem determination in databases [11, 10, 20] as well as enterprise storage systems [25, 28]. Most of these techniques perform diagnosis in an isolated manner attempting to identify root cause(s) of a performance problem in individual database or storage silos. In contrast, DIADS analyzes and correlates data across the database and storage layers.

DB-only Diagnosis: Oracle's Automatic Database Diagnostic Monitor (ADDM) [10, 11] performs fine-grained monitoring to diagnose database performance problems, and to provide tuning recommendations. A similar system [6] has been proposed for Microsoft SQLServer. (Interested readers can refer to [33] for a survey on database problem diagnosis and self-tuning.) However, these tools are oblivious to the underlying SAN layer. They cannot detect problems in the SAN, or identify storage-level root causes that propagate to the database subsystem.

Storage-only Diagnosis: Similarly, there has been research in problem determination and diagnosis in enterprise storage systems. Genesis [25] uses machine learning to identify abnormalities in SANs. A disk I/O throughput model and statistical techniques to diagnose performance problems in the storage layer are described in [28]. There has also been work on profiling techniques for local file systems [3, 36] that help collect data useful in identifying performance bottlenecks as well as in developing models of storage behavior [18, 30, 21].

Drawbacks: Independent database and storage analysis can help diagnose problems like deadlocks or disk failures. However, independent analysis may fail to diagnose problems that do not violate conditions in any one layer, rather contribute cumulatively to the overall poor performance. Two additional drawbacks exist. First, it can involve multiple sets of experts and be time consuming. Second, it may lead to spurious corrective actions as problems in one layer will often surface in another layer. For example, slow I/O due to an incorrect storage volume placement may lead a DB administrator to change the query plan. Conversely, a poor query plan that causes a large number of I/Os may lead the storage administrator to provision more storage bandwidth.

Studies measuring the impact of storage systems on database behavior [27, 26] indicate a strong interdependence between the two subsystems, highlighting the importance of an integrated diagnosis tool like DIADS.

2.2 System Diagnosis Techniques

Diagnosing performance problems has been a popular research topic in the general systems community in recent years [32, 8, 9, 35, 4, 19]. Broadly, this work can be split into two categories: (a) systems using machine learning techniques, and (b) systems using domain knowledge. As described later, DIADS uses a novel mix where machine learning provides the core diagnosis techniques while domain knowledge serves as checks-and-balances against spurious correlations.

Diagnosis based on Machine Learning: PeerPressure [32] uses statistical techniques to develop models for a healthy machine, and uses these models to identify *sick* machines. Another proposed method [4] builds models from process performance counters in order to identify anomalous processes that cause computer slowdowns. There is also work on diagnosing problems in multi-tier Web applications using machine learning techniques. For example, modified Bayesian network models [8] and ensembles of probabilistic models [35] that capture system behavior under changing conditions have been used. These approaches treat data collected from each subsystem equally, in effect creating a single table of performance metrics that is input to machine learning modules. In contrast, DIADS adds more structure and semantics to the collected data, e.g., to better understand the impact of database operator performance vs. SAN volume performance. Furthermore, DIADS complements machine learning techniques with domain knowledge.

Diagnosis based on Domain Knowledge: There are also many systems, especially in the DB community, where

domain knowledge is used to create a *symptoms* database that associates performance symptoms with underlying root causes [34, 19, 24, 10, 11]. Commercial vendors like EMC, IBM, and Oracle use symptom databases for problem diagnosis and correction. While these databases are created manually and require expertise and resources to maintain, recent work attempts to partially automate this process [9, 12].

We believe that a suitable mix of machine learning techniques and domain knowledge is required for a diagnosis tool to be useful in practice. Pure machine learning techniques can be misled by spurious correlations in data resulting from noisy data collection or event propagation (where a problem in one component impacts another component). Such effects need to be addressed using appropriate domain knowledge, e.g., component dependencies, symptoms databases, and knowledge of query plan and operator relationships.

It is also important to differentiate DIADS from tracing-based techniques [7, 1] that trace messages through systems end-to-end to identify performance problems and failures. Such tracing techniques require changes in production system deployments and often add significant overhead in day-to-day operations. In contrast, DIADS performs a postmortem analysis of monitored performance data collected at industry-standard intervals to identify performance problems.

Next, we provide an overview of DIADS.

3 Overview of DIADS

Suppose a query Q that a report-generation application issues periodically to the database system shows a slowdown in performance. One approach to track down the cause is to leverage historic monitoring data collected from the entire system. There are several product offerings [13, 15, 16, 17, 31] in the market that collect and persist monitoring data from IT systems.

DIADS uses a commercial storage management server—IBM TotalStorage Productivity Center [17]—that collects monitoring data from multiple layers of the IT stack including databases, servers, and the SAN. The collected data is transformed into a tabular format, and persisted as time-series data in a relational database.

SAN-level data: The collected data includes: (i) configuration of components (both physical and logical), (ii) connectivity among components, (iii) changes in configuration and connectivity information over time, (iv) performance metrics of components, (v) system-generated events (e.g., disk failure, RAID rebuild) and (vi) events generated by user-defined *triggers* [14] (e.g., degradation in volume performance, high workload on storage subsystem).

Database-level data: To execute a query, a database system generates a *plan* that consists of operators selected from a small, well-defined family of *operators* [14]. Let us consider an example query Q:

```
SELECT   Product.Category, SUM(Product.Sales)
FROM     Product
WHERE    Product.Price > 1000
GROUP BY Product.Category
```

Q asks for the total sales of products, priced above 1000, grouped per category. Figure 1 shows a plan P to execute Q. P consists of four operators: an *Index Scan* of the index on the Price attribute, a *Fetch* to bring matching records from the Product table, a *Sort* to sort these records on Category values, and a *Grouping* to do the grouping and summation. For each execution of P, DIADS collects some monitoring data per operator O. The relevant data includes: O's start time, stop time, and *record-count* (number of records returned in O's output).

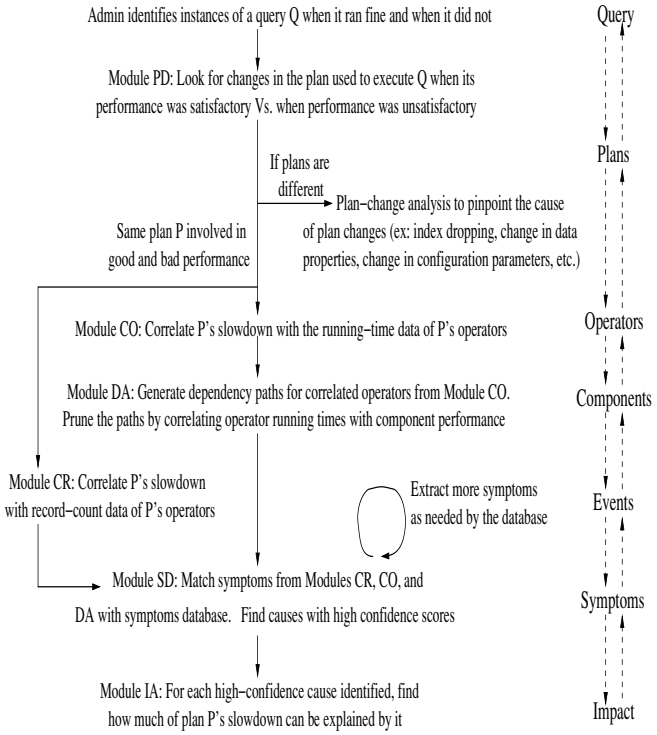

Figure 3: DIADS's diagnosis workflow

DIADS**'s Diagnosis Interface:** DIADS presents an interface where an administrator can mark a query as having experienced a slowdown. Furthermore, the administrator either specifies declaratively or marks directly the runs of the query that were *satisfactory* and those that were *unsatisfactory*. For example, runs with running time below 100 seconds are satisfactory, or all runs between 8 AM and 2 PM were satisfactory, and those between 2 PM and 3 PM were unsatisfactory.

Diagnosis Workflow: DIADS then invokes the *workflow* shown in Figure 3 to diagnose the query slowdown based on the monitoring data collected for satisfactory and unsatisfactory runs. By default, the workflow is run in a

batch mode. However, the administrator can choose to run the workflow in an interactive mode where only one module is run at a time. After seeing the results of each module, the administrator can edit the data or results before feeding them to the next module, bypass or reinvoke modules, or stop the workflow. Because of space constraints, we will not discuss the interactive mode further in this paper.

The first module in the workflow, called Module Plan-Diffing (PD), looks for significant changes between the plans used in satisfactory and unsatisfactory runs. If such changes exist, then DIADS tries to pinpoint the cause of the plan changes (which includes, e.g., index addition or dropping, changes in data properties, or changes in configuration parameters used during plan selection). The techniques used in this module contain details specific to databases, so they are covered in a companion paper [5].

The remaining modules are invoked if DIADS finds a plan P that is involved in both satisfactory and unsatisfactory runs of the query. We give a brief overview before diving into the details in Section 4:

- **Module Correlated Operators (CO):** DIADS finds the (nonempty) subset of operators in P whose change in performance correlates with the query slowdown. The operators in this subset are called *correlated operators*.
- **Module Dependency Analysis (DA):** Having identified the correlated operators, DIADS uses a combination of correlation analysis and the configuration and connectivity information collected during monitoring to identify the components in the system whose performance is correlated with the performance of the correlated operators.
- **Module Correlated Record-counts (CR):** Next, DIADS checks whether the change in P's performance is correlated with the record-counts of P's operators. If significant correlations exist, then it means that data properties have changed between satisfactory and unsatisfactory runs of P.
- **Module Symptoms Database (SD):** The correlations identified so far are likely *symptoms* of the root cause(s) of query slowdown. Other symptoms may be present in the stream of system-generated events and trigger-generated (user-defined) semantic events. The combination of these symptoms is used to probe a *symptoms database* that maps symptoms to the underlying root cause(s). The symptoms database improves diagnosis accuracy by dealing with the propagation of faults across components as well as missing symptoms, unexpected symptoms (e.g., spurious correlations), and multiple simultaneous problems.
- **Module Impact Analysis (IA):** The symptoms database computes a *confidence score* for each suspected root cause. For each high-confidence root cause R, DIADS performs impact analysis to answer the following question: if R is really a cause of the query slowdown, then what fraction of the query slowdown can be attributed to R. To the best of our knowledge, DIADS is the first automated diagnosis tool to have an impact-analysis module.

Integrated database/SAN diagnosis: Note that the workflow "drills down" progressively from the level of the query to plans and to operators, and then uses dependency analysis and the symptoms database to further drill down to the level of performance metrics and events in components. Finally, impact analysis is a "roll up" to tie potential root causes back to their impact on the query slowdown. The drill down and roll up are based on a careful integration of information from the database and SAN layers; and is not a simple concatenation of database-only and SAN-only modules. Only low overhead monitoring data is used in the entire process.

Machine learning + domain knowledge: DIADS's workflow is a novel combination of elements from machine learning with the use of domain knowledge. A number of modules in the workflow use correlation analysis which is implemented using machine learning; the details are in Sections 4.1 and 4.2. Domain knowledge is incorporated into the workflow in Modules DA, SD, and IA; the details are given respectively in Sections 4.2–4.4. (Domain knowledge is also used in Module PD which is beyond the scope of this paper.) As we will demonstrate, the combination of machine learning and domain knowledge provides built-in checks and balances to deal with the challenges listed in Section 1.

4 Modules in the Workflow

We now provide details for all modules in DIADS's diagnosis workflow. Upfront, we would like to point out that our main goal is to describe an end-to-end instantiation of the workflow. We expect that the specific implementation techniques used for the modules will change with time as we gain more experience with DIADS.

4.1 Identifying Correlated Operators

Objective: Given a plan P that is involved in both satisfactory and unsatisfactory runs of the query, DIADS's objective in this module is to find the set of correlated operators. Let O_1, O_2, $\ldots$, O_n be the set of all operators in P. The correlated operators form the subset of $O_1, \ldots, O_n$ whose change in running time best explains the change in P's running time (i.e., P's slowdown).

Technique: DIADS identifies the correlated operators by analyzing the monitoring data collected during satisfactory and unsatisfactory runs of P. This data can be seen as records with attributes $A, t(P), t(O_1), t(O_2), \ldots, t(O_n)$ for each run of P.

Here, attribute $t(P)$ is the total time for one complete run of P, and attribute $t(O_i)$ is the running time of operator O_i for that run. Attribute A is an *annotation* (or *label*) associated with each record that represents whether the corresponding run of P was satisfactory or not. Thus, A takes one of two values: satisfactory (denoted S) or unsatisfactory (denoted U).

Let the values of attribute $t(O_i)$ in records with annotation S be $s_1, s_2, \ldots, s_k$, and those with annotation U be $u_1, u_2, \ldots, u_l$. That is, $s_1, \ldots, s_k$ are k observations of the running time of operator O_i when the plan P ran satisfactorily. Similarly, $u_1, u_2, \ldots, u_l$ are l observations of the running time of O_i when the running time of P was unsatisfactory. DIADS pinpoints correlated operators by characterizing how the distribution of $s_1, \ldots, s_k$ differs from that of $u_1, \ldots, u_l$. For this purpose, DIADS uses *Kernel Density Estimation (KDE)* [22].

KDE is a non-parametric technique to estimate the probability density function of a random variable. Let S_i be the random variable that represents the running time of operator O_i when the overall plan performance is satisfactory. KDE applies a kernel density estimator to the k observations $s_1, \ldots, s_k$ of S_i to learn S_i's probability density function $f_i(S_i)$.

$$f_i(S_i) = \frac{\sum_{j=1}^{k} K(\frac{S_i - s_j}{h})}{kh} \quad (1)$$

Here, K is a *kernel function* and h is a smoothing parameter. A typical kernel is the standard Gaussian function $K(x) = \frac{e^{\frac{-x^2}{2}}}{\sqrt{2\pi}}$. (Intuitively, kernel density estimators are a generalization and improvement over *histograms*.)

Let u be an observation of operator O_i's running time when the plan performance was unsatisfactory. Consider the probability estimate $prob(S_i \leq u) = \int_{-\infty}^{u} f_i(S_i) ds_i$. Intuitively, as u becomes higher than the typical range of values of S_i, $prob(S_i \leq u)$ becomes closer to 1. Thus, a high value of $prob(S_i \leq u)$ represents a significant increase in the running time of operator O_i when plan performance was unsatisfactory compared to that when plan performance was satisfactory.

Specifically, DIADS includes O_i in the set of correlated operators if $prob(S_i \leq \overline{u}) \geq 1 - \alpha$. Here, $\overline{u}$ is the average of $u_1, \ldots, u_l$ and α is a small positive constant. $\alpha = 0.1$ by default. For obvious reasons, $prob(S_i \leq \overline{u})$ is called the *anomaly score* of operator O_i.

4.2 Dependency Analysis

Objective: This module takes the set of correlated operators as input, and finds the set of system components that show a change in performance correlating with the change in running time of one of more correlated operators.

Technique: DIADS implements this module using *dependency analysis* which is based on generating and pruning *dependency paths* for the correlated operators. We describe the generation and pruning of dependency paths in turn.

Generating dependency paths: The dependency path of an operator O_i is the set of physical (e.g., server CPU, database buffer cache, disk) and logical (e.g., volume, external workload) components in the system whose performance can have an impact on O_i's performance. DIADS generates dependency paths automatically based on the following data:

- System-wide configuration and connectivity data as well as updates to this data collected during the execution of each operator (recall Section 3).
- Domain knowledge of how each database operator executes. For example, the dependency path of a sort operator that creates temporary tables on disk will be different from one that does not create temporaries.

We distinguish between *inner* and *outer* dependency paths. The performance of components in O_i's inner dependency path can affect O_i's performance directly. O_i's outer dependency path consists of components that affect O_i's performance indirectly by affecting the performance of components on the inner dependency path. As an example, the inner dependency path for the Index Scan operator in Figure 1 includes the server, HBA, FC-Switches, Pool2, Volume $v2$, and Disks 5-8. The outer dependency path will include Volumes $v1$ and $v3$ (because of the shared disks) and other database queries.

Pruning dependency paths: The fact that a component C is in the dependency path of an operator O_i does not necessarily mean that O_i's performance has been affected by C's performance. After generating the dependency paths conservatively, DIADS prunes these paths based on correlation analysis using KDE.

Recall from Section 3 that the monitoring data collected by DIADS contains multiple observations of the running time of operator O_i both when the overall plan ran satisfactorily and when the plan ran unsatisfactorily. For each run of O_i, consider the performance data collected by DIADS for each component C in O_i's dependency path; this data is collected in the $[t_b, t_e]$ time interval where t_b and t_e are respectively O_i's (absolute) start and stop times for that run. Across all runs, this data can be represented as a table with attributes $A, t(O_i), m_1, \ldots, m_p$. Here, m_1-m_p are performance metrics of component C, and the annotation attribute A represents whether O_i's running time $t(O_i)$ was satisfactory or not in the corresponding run. It follows from Section 4.1 that we can set A's value in a record to U (denoting unsatisfactory) if $prob(S_i \leq t(O_i)) \geq 1 - \alpha$; and to S otherwise.

Given the above annotated performance data for an $\langle O_i, C \rangle$ operator-component pairing, we can apply cor-

	$symp_1$	$symp_2$	$symp_3$	$symp_4$
R_1	1	0	0	1
R_2	1	1	0	0
R_3	1	0	1	1

Figure 4: Example Codebook

relation analysis using KDE to identify C's performance metrics that are correlated with the change in O_i's performance. The details are similar to that in Section 4.1 except for the following: for some performance metrics, observed values lower than the typical range are anomalous. This correlation can be captured using the condition $prob(M \leq v) \leq \alpha$, where M is the random variable corresponding to the metric, v is a value observed for M, and α is a small positive constant.

In effect, the dependency analysis module will identify the set of components that: (i) are part of O_i's dependency path, and (ii) have at least one performance metric that is correlated with the running time of a correlated operator O_i. By default, DIADS will only consider the components in the inner dependency paths of correlated operators. However, components in the outer dependency paths will be considered if required by the symptoms database (Module SD).

Recall Module CR in the diagnosis workflow where DIADS checks for significant correlation between plan P's running time and the record counts of P's operators. DIADS implements this module using KDE in a manner almost similar to the use of KDE in dependency analysis; hence Module CR is not discussed further.

4.3 Symptoms Database

The modules so far in the workflow drilled down from the level of the query to that of physical and logical components in the system; in the process identifying correlated operators and performance metrics. While this information is useful, the detected correlations may only be *symptoms* of the true root cause(s) of the query slowdown. This issue, which can mask the true root cause(s), is generally referred to as the *event (fault) propagation* problem in diagnosis. For example, a change in data properties at the database level may, in turn, propagate to the volume level causing volume contention, and to the server level increasing CPU utilization. In addition, some spurious correlations may creep in and manifest themselves as unexpected symptoms in spite of our careful drill down process.

Objective: DIADS's Module SD tries to map the observed symptoms to the actual root cause(s), while dealing with missing as well as unexpected symptoms arising from the noisy nature of production systems.

Technique: DIADS uses a *symptoms database* to do the mapping. This database streamlines the use of domain knowledge in the diagnosis workflow to:

- Generate more accurate diagnosis results by dealing with event propagation.
- Generate diagnosis results that are semantically more meaningful to administrators (for example, reporting lock contention as the root cause instead of reporting some correlated metrics only).

We considered a number of formats proposed previously in the literature to input domain knowledge for aiding diagnosis. Our evaluation criteria were the following:

I. How easy is the format for administrators to use? Here, usage includes customization, maintenance over time, as well as debugging. When a diagnosis tool pinpoints a particular cause, it is important that the administrators are able to understand and validate the tool's reasoning. Otherwise, administrators may never trust the tool enough to use it.

II. Can the format deal with the noisy conditions in production systems, including multiple simultaneous problems, presence of spurious correlations, and missing symptoms.

One of the formats from the literature [16] is an *expert knowledge-base* of rules where each rule expresses patterns or relationships that describe symptoms, and can be matched against the monitoring data. Most of the focus in this work has been on exact matches, so this format scores poorly on Criterion II. Representing relationships among symptoms (e.g., event X will cause event Y) using deterministic or probabilistic networks like Bayesian networks [23] has been gaining currency recently. This format has high expressive power, but remains a black-box for administrators who find it hard to interpret the reasoning process (Criterion I).

Another format, called the *Codebook* [34], is very intuitive as well as implemented in a commercial product. This format assumes a finite set of symptoms such that each distinct root cause R has a unique *signature* in this set. That is, there is a unique subset of symptoms that R gives rise to which differs makes it distinguishable from all other root causes. This information is represented in the Codebook which is a matrix whose columns correspond to the symptoms and rows correspond to the root causes. A cell is mapped to 1 if the corresponding root cause should show the corresponding symptom; and to 0 otherwise. Figure 4 shows an example Codebook where there are four hypothetical symptoms $symp_1$–$symp_4$ and three root causes R_1–R_3.

When presented with a vector V of symptoms seen in the system, the Codebook computes the *distance* $d(V, R)$ of V to each row R (i.e., root cause). Any number of different distance metrics can be used, e.g., Euclidean (L_2) distance or Hamming distance [34]. $d(V, R)$ is a measure of the confidence that R is a root cause of the problem. For example, given a symptoms vector $\langle 1, 0, 0, 1 \rangle$ (i.e., only $symp_1$ and $symp_4$ are seen), the Euclidean

distances to the three root causes in Figure 4 are 0, $\sqrt{2}$, and 1 respectively. Hence, R_1 is the best match.

The Codebook format does well on both our evaluation criteria. Codebooks can handle noisy situations, and administrators can easily validate the reasoning process. However, DIADS needs to consider complex symptoms such as symptoms with temporal properties. For example, we may need to specify a symptom where a disk failure is seen within X minutes of the first incidence of the query slowdown, where X may vary depending on the installation. Thus, it is almost impossible in our domain to fully enumerate a closed space of relevant symptoms, and to specify for each root cause whether each symptom from this space will be seen or not. These observations led to DIADS's new design of the symptoms database:

1. We define a *base set* of symptoms consisting of: (i) operators in the database system that can be included in the correlated set, (ii) performance metrics of components that can be correlated with operator performance, and (iii) system-monitored and user-defined events collected by DIADS.

2. The language defined by IBM's *Active Correlation Technology (ACT)* is used to express complex symptoms over the base set of symptoms [2]. The benefit of this language comes from its support for a range of built-in patterns including filter, collection, duplicate, computation, threshold, sequence, and timer. ACT can express symptoms like: (i) the workload on a volume is higher than 200 IOPS, and (ii) event E_1 should follow event E_2 in the 30 minutes preceding the first instance of query slowdown.

3. DIADS's symptoms database is a collection of root cause entries each of which has the format $Cond_1$ & $Cond_2$ & ... & $Cond_z$, for some $z > 0$ which can differ across entries. Each $Cond_i$ is a Boolean condition of the form $\exists symp_j$ (denoting presence of $symp_j$) or $\neg\exists symp_j$ (denoting absence of $symp_j$). Here, $symp_j$ is some base or complex symptom. Each $Cond_i$ is associated with a weight w_i such the sum of the weights for each individual root cause entry is 100%. That is, $\sum_{i=1}^{z} w_i = 100\%$.

4. Given a vector of base symptoms, DIADS computes a *confidence score* for each root cause entry R as the sum of the weights of R's conditions that evaluate to true. Thus, the confidence score for R is a value in $[0\%, 100\%]$ equal to $\sum_{i=1}^{z} w_i | Cond_i = true$.

DIADS's symptoms database tries to balance the expressive power of rules with the intuitive structure and robustness of Codebooks. The symptoms database differs from conventional Codebooks in a number of ways. For each root cause entry, DIADS avoids the "closed-world" assumption for symptoms by mapping symptoms to 0, 1, or "don't care". Conventional Codebooks are constrained to 0 or 1 mappings. DIADS's symptoms database can contain mappings for *fixes* to problems in addition to root causes. This feature is useful because it may be easier to specify a fix for a query slowdown (e.g., add an index) instead of trying to find the root cause. DIADS also allows multiple distinct entries for the same root cause.

Generation of the symptoms database: Companies like EMC, IBM, HP, and Oracle are investing significant (currently, mostly manual) effort to create symptoms databases for different subsystems like networking infrastructure, application servers, and databases [34, 19, 24, 9, 10, 11]. Symptoms databases created by some of these efforts are already in commercial use. The creation of these databases can be partially automated, e.g., through a combination of fault injection and machine learning [9, 12]. In fact, DIADS's modules like correlation, dependency, and impact analysis can be used to identify important symptoms automatically.

4.4 Impact Analysis

Objective: The confidence score computed by the symptoms database module for a potential root cause R captures how well the symptoms seen in the system match the expected symptoms of R. For each root cause R whose confidence score exceeds a threshold, the impact analysis module computes R's *impact score*. If R is an actual root cause, then R's impact score represents the fraction of the query slowdown that can be attributed to R individually. DIADS's novel impact analysis module serves three significant purposes:

- When multiple problems coexist in the system, impact analysis can separate out high-impact causes from the less significant ones; enabling prioritization of administrator effort in problem solving.
- As a safeguard against misdiagnoses caused by spurious correlations due to noise.
- As an extra check to find whether we have identified the right cause(s) or all cause(s).

Technique: Interestingly, one approach for impact analysis is to *invert* the process of dependency analysis from Section 4.2. Let R be a potential root cause whose impact score needs to be estimated:

1. Identify the set of components, denoted $comp(R)$, that R affects in the inner dependency path of the operators in the query plan. DIADS gets this information from the symptoms database.
2. For each component $C \in comp(R)$, find the subset of correlated operators, denoted $op(R)$, such that for each operator O in this subset: (i) C is in O's inner dependency path, and (ii) at least one performance metric of C is correlated with the change in

O's performance. DIADS has already computed this information in the dependency analysis module.

3. R's impact score is the percentage of the change in plan running time (query slowdown) that can be attributed to the change in running time of operators in $op(R)$. Here, change in running time is computed as the difference between the average running times when performance is unsatisfactory and that when performance is satisfactory.

The above approach will work as long as for any pair of suspected root causes R_1 and R_2, $op(R_1) \cap op(R_2) = \emptyset$. However, if there are one or more operators common to $op(R_1)$ and $op(R_2)$ whose running times have changed significantly, then the above approach cannot fully separate out the individual impacts of R_1 and R_2.

DIADS addresses the above problem by leveraging *plan cost models* that play a critical role in all database systems. For each query submitted to a database system, the system will consider a number of different plans, use the plan cost model to predict the running time (or some other cost metric) of each plan, and then select the plan with minimum predicted running time to run the query to completion. These cost models have two main components:

- Analytical formula per operator type (e.g., sort, index scan) that estimates the resource usage (e.g., CPU and I/O) of the operator based on the values of input parameters. While the number and types of input parameters depend on the operator type, the main ones are the sizes of the input processed by the operator.
- Mapping parameters that convert resource-usage estimates into running-time estimates. For example, IBM DB2 uses two such parameters to convert the number of estimated I/Os into a running-time estimate: (i) the overhead per I/O operation, and (ii) the transfer rate of the underlying storage device.

The following are two examples of how DIADS uses plan cost models:

- Since DIADS collects the old and new record-counts for each operator, it estimates the impact score of a change in data properties by plugging the new record-counts into the plan cost model.
- When volume contention is caused by an external workload, DIADS estimates the new I/O latency of the volume from actual observations or the use of device performance models. The impact score of the volume contention is computed by plugging this new estimate into the plan cost model.

DIADS's use of plan cost models is a general technique for impact analysis, but it is limited by what effects are accounted for in the model. For example, if wait times for locks are not modeled, then the impact score cannot be computed for locking-based problems. Addressing this issue—e.g., by extending plan cost models or by using *planned experiments* at run time—is an interesting avenue for future work.

5 Experimental Evaluation

The taxonomy of scenarios considered for diagnosis in the evaluation follows from Figure 2. DIADS was used to diagnose query slowdowns caused by (i) events within the database and the SAN layers, (ii) combinations of events across both layers, as well as (iii) multiple concurrent problems (a capability unique to DIADS). Due to space limitations, it is not possible to describe all the scenario permutations from Figure 2. Instead, we start with a scenario and make it increasingly complex by combining events across the database and SAN. We consider: (i) volume contention caused by SAN misconfiguration, (ii) database-level problems (change in data properties, contention due to table locking) whose symptoms propagate to the SAN, and (iii) independent and concurrent database-level and SAN-level problems.

We provide insights into how DIADS diagnoses these problems by drilling down to the intermediate results like anomaly, confidence, and impact scores. While there is no equivalent tool available for comparison with DIADS, we provide insights on the results that a database-only or SAN-only tool would have generated; these insights are derived from hands-on experience with multiple in-house and commercial tools used by administrators today. Within the context of the scenarios, we also report sensitivity analysis of the anomaly score to the number of historic samples and length of the monitoring interval.

5.1 Setup Details

Our experimental testbed is part of a production SAN environment, with the interconnecting fabric and storage controllers being shared by other applications. Our experiments ran during low activity time-periods on the production environment. The testbed runs data-warehousing queries from the popular TPC-H benchmark [29] on a PostgreSQL database server configured to access tables using two Ext3 filesystem volumes created on an enterprise-class IBM DS6000 storage controller. The database server is a 2-way 1.7 GHz IBM xSeries machine running Linux (Redhat 4.0 Server), connected to the storage controller via Fibre Channel (FC) host bus adaptor (HBA). Both the storage volumes are RAID 5 configurations consisting of (4 + 2P) 15K FC disks.

An IBM TotalStorage Productivity Center [17] SAN management server runs on a separate machine recording configuration details, statistics, and events from the SAN as well as from PostgreSQL (which was instrumented to report the data to the management tool). Figure 6 shows the key performance metrics collected from the database and SAN. The monitoring data is stored as time-series data in a DB2 database. Each module in DI-

ADS's workflow is implemented using a combination of Matlab scripts (for KDE) and Java. DIADS uses a symptoms database that was developed in-house to diagnose query slowdowns in database over SAN deployments.

Our experimental results focus on the slowdown of the plan shown in Figure 5 for Query 2 from TPC-H. Figure 5 shows the 25 operators in the plan, denoted O_1–O_{25}. In database terminology, the operators Index Scan and Sequential Scan are *leaf* operators since they access data directly from the tables; hence the leaf operators are the most sensitive to changes in SAN performance. The plan has 9 leaf operators. The other operators process intermediate results.

5.2 Scenario 1: Volume Contention due to SAN Misconfiguration

Problem Setting

In this scenario, a contention is created in volume V1 (from Figure 5) causing a slowdown in query performance. The root cause of the contention is another application workload that is configured in the SAN to use a volume V' that gets mapped to the same physical disks as V1. For an accurate diagnosis result, DIADS needs to pinpoint the combination of SAN configuration events generated on: (i) creation of the new volume V', and (ii) creation of a new zoning and mapping relationship of the server running the workload that accesses V'.

Module CO

DIADS analyzes the historic monitoring samples collected for each of the 25 query operators. The monitoring samples for an operator are labeled as satisfactory or unsatisfactory based on past problem reports from the administrator. Using the operator running times in these labeled samples, Module CO in the workflow uses KDE to compute anomaly scores for the operators (recall Section 4.1). Table 1 shows the anomaly scores of the operators identified as the correlated operators; these operators have anomaly scores ≥ 0.8 (the significance of the anomaly scores is covered in Section 4.1). The following observations can be made from Table 1:

- Leaf operators O_8 and O_{22} were correctly identified as correlated. These two are the only leaf operators that access data on the Volume V1 under contention.
- Eight intermediate operators were ranked highly as well. This ranking can be explained by event propagation where the running times of these operators are affected by the running times of the "upstream" operators in the plan (in this case O_8 and O_{22}).
- A false positive for leaf operator O_4 which operates on tables in Volume V2. This could be a result of noisy monitoring data associated with the operator.

In summary, Module CO's KDE analysis has zero false negatives and one false positive from the total set of 9

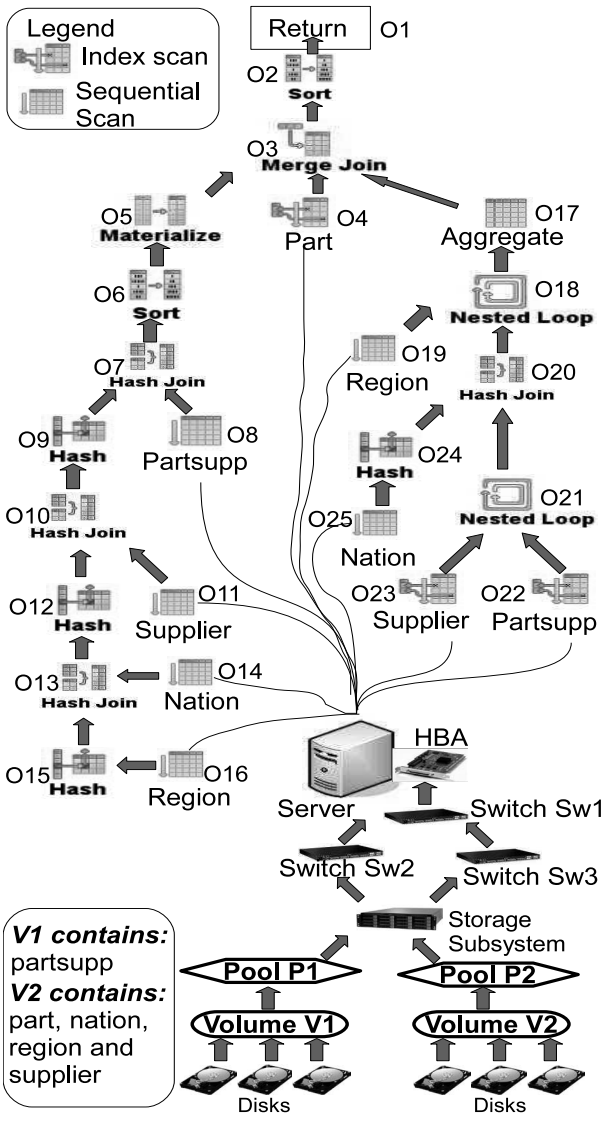

Figure 5: Query plan, operators, and dependency paths for the experimental results

leaf operators. The false positive gets filtered out later in the symptoms database and impact analysis modules.

To further understand the anomaly scores, we conducted a series of sensitivity tests. Figure 7 shows the sensitivity of the anomaly scores of three representative operators to the number of samples available from the satisfactory runs. O_{22}'s score converges quickly to 1 because O_{22}'s running time under volume contention is almost 5X the normal. However, the scores for leaf operator O_{11} and intermediate operator O_1 take around 20 samples to converge. With fewer than these many samples, O_{11} could have become a false positive. In all our results, the anomaly scores of all 25 operators converge within 20 samples. While more samples may be required in environments with higher noise levels, the relative simplicity of KDE (compared to models like Bayesian

Database Metrics	Server Metrics	Network Metrics	Storage Metrics
Operator Start Stop Times	CPU Usage (%ge)		
Record-counts	CPU Usage (Mhz)	Bytes Transmitted	Bytes Read
Plan Start Stop Times	Handles	Bytes Received	Bytes Written
Locks outstanding and held	Threads	Packets Transmitted	Contaminating Writes
	Processes	Packets Received	PhysicalStorageRead Operations
Lock wait times	Heap Memory Usage(KB)	LIP Count	Physical Storage Read Time
Space Usage	Physical Memory Usage (%)	NOS Count	PhysicalStorageWriteOperations
Blocks Read	Kernel Memory(KB)	Error Frames	Physical Storage Write Time
Buffer Hits	Memory Being Swapped(KB)	Dumped Frames	Sequential Read Requests
Index Scans	Reserved Memory	Link Failures	Sequential Write Requests
Index Reads	Capacity(KB)	CRC Errors	Total IOs
Index Fetches	Wait I/O	Address Errors	
Sequential Scans	Network Bandwidth (HBA)		

Figure 6: Important performance metrics collected by DIADS

Operator	Operator Type	Anomaly Score
O_2	Non-leaf	1.00
O_3	Non-leaf	1.00
O_6	Non-leaf	1.00
O_7	Non-leaf	1.00
O_8	Leaf (sequential scan)	1.00
O_{18}	Non-leaf	1.00
O_{20}	Non-leaf	1.00
O_{21}	Non-leaf	1.00
O_{22}	Leaf (index scan)	1.00
O_{17}	Non-leaf	0.969
O_4	Leaf (index scan)	0.965

Table 1: Anomaly scores for query operators from Figure 5 in Scenario 1

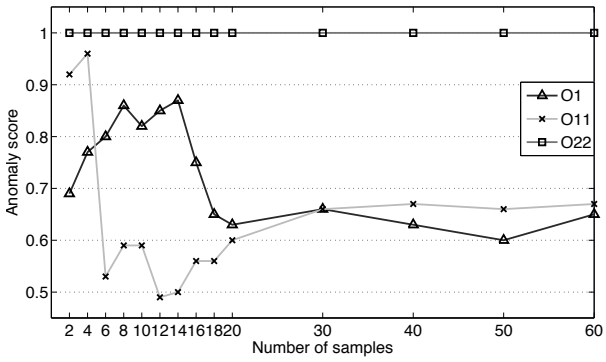

Figure 7: Sensitivity of anomaly scores to the number of satisfactory samples. While O_{22} shows highly anomalous behavior, scores for O_1 and O_{11} should be low

networks) keeps this number low.

Figure 8 shows the sensitivity of O_{22}'s anomaly score to the length of the monitoring interval during a 4-hour period. Intuitively, larger monitoring intervals suppress the effect of spikes and bursty access patterns. In our experiments, the query running time was around 4 minutes under satisfactory conditions. Thus, monitoring intervals of 10 minutes and larger in Figure 8 cause the anomaly score to deviate more and more from the true value.

Module DA

This module generates and prunes dependency paths for correlated operators in order to relate operator performance to database and SAN component performance. For ease of presentation, we will focus on the leaf operators in Figure 5 since they are the most sensitive to SAN performance. Given the configuration of our experimental testbed in Figure 5, the primary difference between the dependency paths of various operators is in the volumes they access: V1 is in the dependency path of O_8 and O_{22}, and V2 is in the paths of O_4, O_{11}, O_{14}, O_{16}, O_{19}, O_{23}, and O_{25}.

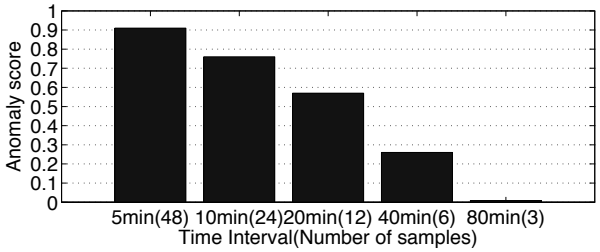

Figure 8: Sensitivity of anomaly scores to noise in the monitoring data

Volume, Perf. Metric	Anomaly Score (no contention in V2)	Anomaly Score (contention in V2)
V1, writeIO	0.894	0.894
V1, writeTime	0.823	0.823
V2, writeIO	0.063	0.512
V2, writeTime	0.479	0.879

Table 2: Anomaly scores computed during dependency analysis for performance metrics from Volumes V1, V2

The set of correlated operators from Module CO are O_4, O_8, and O_{22}. Thus, DIADS will compute anomaly scores for the performance metrics of both V1 and V2. Table 2's second column shows the anomaly scores for two representative metrics each from V1 and V2. (Table 2's third column is described later in this section.) As expected, none of V2's metrics are identified as correlated because V2 has no contention; while those of V1 are.

Module CR

Anomaly scores are low in this module because data properties do not change.

Module SD

The symptoms identified up to this stage are:

- High anomaly scores for operators dependent on V1.
- High anomaly scores for V1's performance metrics.
- High anomaly score for only one V2-dependent operator (out of seven such operators).

These symptoms are strong evidence that V1's performance is a cause of the query slowdown, and V2's performance is not. Thus, even when a symptoms database is not available, DIADS correctly narrows down the search space an administrator has to consider during diagnosis. An impact analysis will further point out that the false positive symptom due to O_4 has little impact on the query slowdown.

However, without a symptoms database or further diagnosis effort from the administrator, the *root cause* of V1's change of performance is still unknown among possible candidates like: (i) change of performance of an external workload, (ii) a runaway query in the database, or (iii) a RAID rebuild. We will now report results from the use of a symptoms database that was developed in-house. DIADS uses this database as described in Section 4.3 except that instead of reporting numeric confidence scores to administrators, DIADS reports confidence as one of High ($score \geq 80\%$), Medium ($80\% > score \geq 50\%$), or Low ($50\% > score \geq 0\%$). The summary of Module SD's output in the current scenario is:

- All root causes with contention-related symptoms for V2 have Low confidence (few symptoms are found).
- RAID rebuild gets Low confidence because no RAID rebuild start or end events are found.
- V1 contention due to changes in data properties gets Low confidence because symptoms are missing.
- V1 contention due to change in external workload gets Low confidence because no external workload was on the outer dependency path of a correlated operator when performance was satisfactory.
- V1 contention due to change in database workload gets Medium confidence because of a weak correlation between the performance of some correlated operators and the rest of the database workload.
- V1 contention due to the SAN misconfiguration problem gets High confidence because all specified symptoms are found including: (i) creation of a new volume (parametrized with the physical disk information), and (ii) creation of new masking and zoning information for the volume.

The symptoms database had an entry for the actual root cause because this problem is common. Hence, DIADS was able to diagnose the root cause for this scenario. Note that DIADS had to consider more than 900 events (system generated as well as user-defined) for the database and SAN generated during the course of the satisfactory and unsatisfactory runs for this experiment.

Module IA

Impact analysis done using the inverse dependency analysis technique gives an impact score of 99.8% for the high-confidence root cause found. This score is high because the slowdown is caused entirely by the contention in V1.

In keeping with our experimental methodology, we complicated the problem scenario to test DIADS's robustness. Everything was kept the same except that we created extra I/O load on Volume V2 in a bursty manner such that this extra load had little impact on the query beyond the original impact of V1's contention. Without intrusive tracing, it would not be possible to rule out the extra load on V2 as a potential cause of the slowdown.

Interestingly, DIADS's integrated approach is still able to give the right answer. Compared to the previous scenario, there will now be some extra symptoms due to higher anomaly scores for V2's performance metrics (as shown in the third column in Table 2). However, root causes with contention-related symptoms for V2 will still have Low confidence because most of the leaf operators depending on V2 will have low anomaly scores as before. Also, impact scores will be low for these causes.

Unlike DIADS, a SAN-only diagnosis tool may spot higher I/O loads in both V1 and V2, and attribute both of these as potential root causes. Even worse, the tool may give more importance to V2 because most of the data is on V2. A database-only tool can pinpoint the slowdown in the operators. However, this tool cannot track the root cause down to the SAN level because it has no visibility into SAN configuration or performance. From our experience, database-only tools may give several false positives in this context, e.g., suboptimal bufferpool setting or a suboptimal choice of execution plan.

5.3 Scenario 2: Database-layer Problem Propagating to the SAN-layer

In this scenario we cause a query slowdown by changing the properties of the data, causing extra I/O on Volume V2. The change is done by an update statement that modifies the value of an attribute in some records of the *part* table. The overall size of all tables, including *part*, are unchanged. There are no external causes of contention on the volumes.

Modules CO, DA, and CR behave as expected. In particular, module CR correctly identifies all the operators whose record-counts show a correlation with plan performance: operators O_1, O_2, O_3, and O_4 show increased record-counts, while operators O_5 and O_6 show reduced record-counts. The root-cause entry for changes in data properties gets High confidence in Module SD because all needed symptoms match. All other root-cause entries

get Low confidence, including contention due to changes in external workload and database workload because no correlations are detected on the outer dependency paths of correlated operators (as expected).

The impact analysis module gives the final confirmation that the change in data properties is the root cause, and rules out the presence of high-impact external causes of volume contention. As described in Section 4.4, we can use the plan cost model from the database to estimate the individual impact of any change in data properties. In this case, the impact score for the change in data properties is 88.31%. Hence, DIADS could have diagnosed the root cause of this problem even if the symptoms database was unavailable or incomplete.

5.4 Scenario 3: Concurrent Database-layer and SAN-layer Problems

We complicate Scenario 2 by injecting contention on Volume V2 due to SAN misconfiguration along with the change in data properties. Both these problems individually cause contention in V2. The SAN misconfiguration is the higher-impact cause in our testbed. This key scenario represents the occurrence of multiple, possibly related, events at the database and SAN layers, complicating the diagnosis process. The expected result from DIADS is the ability to pinpoint both these events as causes, and giving the relative impact of each cause on query performance.

The CO, DA, and CR Modules behave in a fashion similar to Scenario 2, and drill down to the contention in Volume V2. We considered DIADS's performance in two cases: with and without the symptoms database. When the symptoms database is unavailable or incomplete, DIADS cannot distinguish between Scenarios 2 and 3. However, DIADS's impact analysis module computes the impact score for the change in data properties, which comes to 0.56%. (This low score is representative because the SAN misconfiguration has more than 10X higher impact on the query performance than the change in data properties.) Hence, DIADS final answer in this case is as follows: (i) a change in data properties is a high-confidence but low-impact cause of the problem, and (ii) there are one or more other causes that impact V2 which could not be diagnosed.

When the symptoms database is present, both the actual root causes are given High confidence by Module SD because the needed symptoms are seen in both cases. Thus, DIADS will pinpoint both the causes. Furthermore, impact analysis will confirm that the full impact on the query performance can be explained by these two causes.

A database-only diagnosis tool would have successfully diagnosed the change in data properties in both Scenarios 2 and 3. However, the tool may have difficulty distinguishing between these two scenarios or pinpointing causes at the SAN layer. A SAN-only diagnosis tool will pinpoint the volume overload. However, it will not be able to separate out the impacts of the two causes. Since the sizes of the tables do not change, we also suspect that such a tool may even rule out the possibility of a change in data properties being a cause.

5.5 Discussion

The scenarios described in the experimental evaluation were carefully chosen to be simple, but not simplistic. They are representative of event categories occurring within the DB and SAN layers as shown in Figure 2. We have additionally experimented with different events within those categories such as CPU and memory contention in the SAN in addition to disk-level saturation, different types of database misconfiguration, and locking-based database problems. Locking-based problems are hard to diagnose because they can cause different types of symptoms in the SAN layer, including contention as well as underutilization. We have also considered concurrent occurrence of three or more problems, e.g., change in data properties, SAN misconfiguration, and locking-based problems. The insights from these experiments are similar to those seen already, and further confirm the utility of an integrated tool. However:

- High levels of noise in the monitoring data can reduce DIADS's effectiveness.
- While DIADS would still be effective when the symptoms database is incomplete, more manual effort will be needed to pinpoint actual root causes.
- Incomplete or inaccurate plan cost models reduce the accuracy of impact analysis.

6 Conclusions and Future Work

We presented an integrated database and storage diagnosis tool called DIADS. Using a novel combination of machine learning techniques with database and storage expert domain-knowledge, DIADS accurately identifies the root cause(s) of problems in query performance; irrespective of whether the problem occurs in the database or the storage layer. This integration enables a more accurate and efficient diagnosis tool for system administrators. Through a detailed experimental evaluation, we also demonstrated the robustness of our approach: with its ability to deal with concurrent multiple problems as well as presence of noisy data.

In future, we are interested in exploring two directions of research. First, we are investigating approaches that further strengthen the analysis done as part of DIADS modules, e.g., techniques that complement database query plan models using planned run-time experiments. Second, we aim to generalize our diagnosis techniques to support applications other than databases in conjunction with enterprise storage.

References

[1] AGUILERA, M. K., MOGUL, J. C., WIENER, J. L., REYNOLDS, P., AND MUTHITACHAROEN, A. Performance Debugging for Distributed Systems of Black Boxes. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)* (2003), pp. 74–89.

[2] ANA BIAZETTI AND KIM GAJDA. *Achieving Complex Event Processing with Active Correlation Technology*. http://www.ibm.com/developerworks/library/ac-acact/index.html.

[3] ARANYA, A., WRIGHT, C. P., AND ZADOK, E. Tracefs: A File System to Trace Them All. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST)* (2004), pp. 129–145.

[4] BASU, S., DUNAGAN, J., AND SMITH, G. Why Did My PC Suddenly Slow Down? In *Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques (SYSML)* (2007), pp. 1–6.

[5] BORISOV, N., UTTAMCHANDANI, S., ROUTRAY, R., AND SINGH, A. Why Did My Query Slow Down? In *Proceedings of the Fourth Biennial Conference on Innovative Data Systems Research (CIDR)* (2009).

[6] CHAUDHURI, S., KÖNIG, A. C., AND NARASAYYA, V. R. SQLCM: A Continuous Monitoring Framework for Relational Database Engines. In *Proceedings of the International Conference on Data Engineering (ICDE)* (2004), pp. 473–485.

[7] CHEN, M. Y., ACCARDI, A., KICIMAN, E., LLOYD, J., PATTERSON, D., FOX, A., AND BREWER, E. Path-based Faliure and Evolution Management. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)* (2004), pp. 23–23.

[8] COHEN, I., CHASE, J. S., GOLDSZMIDT, M., KELLY, T., AND SYMONS, J. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. In *Proceedings of the USENIX Symp. on Operating Systems Design and Implementation (OSDI)* (Dec. 2004), pp. 104–109.

[9] COHEN, I., ZHANG, S., GOLDSZMIDT, M., SYMONS, J., KELLY, T., AND FOX, A. Capturing, Indexing, Clustering, and Retrieving System History. In *ACM symposium on Operating systems principles (SOSP)* (2005), pp. 105–118.

[10] DAGEVILLE, B., DAS, D., DIAS, K., YAGOUB, K., ZAIT, M., AND ZIAUDDIN, M. Automatic SQL Tuning in Oracle 10g. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)* (2004), pp. 1098–1109.

[11] DIAS, K., RAMACHER, M., SHAFT, U., VENKATARAMANI, V., AND WOOD, G. Automatic Performance Diagnosis and Tuning in Oracle. In *Proceedings of Conference on Innovative Data Systems Research (CIDR)* (2005), pp. 84–94.

[12] DUAN, S., BABU, S., AND MUNAGALA, K. Fa: A System for Automating Failure Diagnosis. In *Proceedings of the International Conference on Data Engineering (ICDE)* (Apr. 2009).

[13] EMC CONTROL CENTER FAMILY. http://www.emc.com/products/storage_management/controlcenter.jsp.

[14] GARCIA-MOLINA, H., ULLMAN, J., AND WIDOM, J. *Database Systems: The Complete Book*. Prentice Hall, Upper Saddle River, New Jersey, 2001.

[15] HEWLETT PACKARD SYSTEMS INSIGHT MANAGER. http://h18002.www1.hp.com/products/servers/management/hpsim/index.html.

[16] IBM TIVOLI NETWORK MANAGER. http://www-01.ibm.com/software/tivoli/products/netcool-precision-ip.

[17] IBM TOTALSTORAGE PRODUCTIVITY CENTER. http://www-306.ibm.com/software/tivoli/products/totalstorage-data/.

[18] JOUKOV, N., TRAEGER, A., IYER, R., WRIGHT, C. P., AND ZADOK, E. Operating System Profiling via Latency Analysis. In *Proceedings of the 7th symposium on Operating systems design and implementation (OSDI)* (2006), pp. 89–102.

[19] MANJI, A. Creating Symptom Databases to Rervice J2EE Applications in WebSphere Studio, 2004.

[20] MEHTA, A., GUPTA, C., WANG, S., AND DAYAL, U. Automatic Workload Management for Enterprise Data Warehouses. *IEEE Data Eng. Bull. 31*, 1 (2008), 11–19.

[21] MESNIER, M. P., WACHS, M., SAMBASIVAN, R. R., ZHENG, A. X., AND GANGER, G. R. Modeling the Relative Fitness of Storage. *SIGMETRICS Perform. Eval. Rev. 35*, 1 (2007), 37–48.

[22] PARZEN, E. On Estimation of a Probability Density Function and Mode. *Ann. Math. Stat. 33* (1962), 1065–1076.

[23] PEARL, J. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, Cambridge, UK, 2000.

[24] PERAZOLO, M. The Autonomic Computing Symptoms Format, 2005. IBM Library.

[25] POLLACK, K. T., AND UTTAMCHANDANI, S. Genesis: A Scalable Self-Evolving Performance Management Framework for Storage Systems. In *IEEE International Conference on Distributed Computing Systems (ICDCS)* (2006), p. 33.

[26] QIN, Y., SALEM, K., AND GOEL, A. K. Towards Adaptive Costing of Database Access Methods. In *Proceedings of the International Workshop on Self-Managing Database Systems (SMDB)* (2007), pp. 469–477.

[27] REISS, F., AND KANUNGO, T. A Characterization of the Sensitivity of Query Optimization to Storage Access Cost Parameters. In *SIGMOD Conference* (2003), pp. 385–396.

[28] SHEN, K., ZHONG, M., AND LI, C. I/O System Performance Debugging Using Model-driven Anomaly Characterization. In *Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies (FAST)* (2005), pp. 23–23.

[29] THE TPC-H DECISION SUPPORT BENCHMARK. www.tpc.org/tpch.

[30] THERESKA, E., SALMON, B., STRUNK, J., WACHS, M., ABD-EL-MALEK, M., LOPEZ, J., AND GANGER, G. R. Stardust: Tracking Activity in a Distributed Storage System. *SIGMETRICS Perform. Eval. Rev. 34*, 1 (2006), 3–14.

[31] VMWARE VIRTUAL CENTER. http://www.vmware.com/products/vi/vc/.

[32] WANG, H. J., PLATT, J. C., CHEN, Y., ZHANG, R., AND WANG, Y.-M. Automatic Misconfiguration Troubleshooting with PeerPressure. In *Proceedings of USENIX OSDI Conference* (2004), pp. 245–258.

[33] WEIKUM, G., MOENKEBERG, A., HASSE, C., AND ZABBACK, P. Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering. In *Proceedings of the VLDB Conference* (Hongkong, China, 2002), pp. 20–31.

[34] YEMINI, S. A., KLIGER, S., MOZES, E., YEMINI, Y., AND OHSIE, D. High Speed and Robust Event Correlation. *IEEE Communications Magazine 34* (1996).

[35] ZHANG, S., COHEN, I., SYMONS, J., AND FOX, A. Ensembles of Models for Automated Diagnosis of System Performance Problems. In *Proceedings of International Conference on Dependable Systems and Networks (DSN)* (2005), pp. 644–653.

[36] ZHOU, S., COSTA, H. D., AND SMITH, A. J. A File System Tracing Package for Berkeley UNIX. Tech. rep., 1985.