

# Why Did My Query Slow Down?

Nedyalko Borisov<sup>†</sup>

Sandeep Uttamchandani<sup>‡</sup>

Ramani Routray<sup>‡</sup>

Aameek Singh<sup>‡</sup>

<sup>†</sup>Duke University

<sup>‡</sup>IBM Almaden Research Center

nedyalko@cs.duke.edu

{sandeepu, routrayr, aameek.singh}@us.ibm.com

## ABSTRACT

Enterprise environments have isolated teams responsible separately for database management and the management of underlying network-attached server-storage infrastructure (referred to as Storage Area Networks or SANs). Our vision is to develop an innovative management framework to simplify administrative tasks that require an understanding of both database and SAN details. As a concrete instance, this paper describes problem diagnosis of a database query slowdown. To address the diagnosis challenges including noisy monitoring data, we have implemented DIADS, which uses existing monitoring tools to generate *Annotated Plan Graphs (APGs)*. Using an innovative workflow that combines domain-specific knowledge with machine-learning, DIADS was successfully applied on a real-world testbed to pin-point complex combination of events across both the database and SAN.

## 1. INTRODUCTION

In enterprise environments, databases support high transaction rates and operate on terabyte scale data footprints. Traditionally, storage sub-systems were directly attached to high-end database servers to meet their capacity, throughput, and bandwidth requirements. Economic realities of high administration costs for islands of disconnected resources combined with under-utilization of statically provisioned server and storage hardware have transformed the direct-attached architectures into a network-attached setup with multiple application servers (including databases) connected to a consolidated and virtualized storage pool, an architecture popularly known as a Storage Area Network (SAN). A typical SAN has a hierarchy of *core* and *edge* fibre-channel switches with *zoning* configuration that controls connectivity of incoming server ports with one or more heterogeneous storage controllers which has disks carved into logical entities of ranks, pools, and volumes. Given these complexities, database administrators treat the SAN as a black-box, entrusting SAN administrators for configuring the required CPU, network and storage resources for meeting their performance requirements. While a isolated approach for database and SAN management is the state-of-art today, it is becoming increasingly infeasible for several administrative tasks namely optimized allocation of SAN resources for varying database workload characteristics, diagnosis of problems related to query performance slowdown, what-if analysis to evaluate database performance impact as a result of deploying new application workloads in the SAN. As a concrete instance, this paper focuses on integrated problem diagnosis of database and SAN for query performance slowdowns.

Consider a real-world problem diagnosis scenario. In the event of query slowdown, the database administrator opens a problem ticket for the SAN administrator to analyze and fix: “*The online transaction processing database myOLTP has a 30% slow down in processing time, compared to performance two weeks back.*” 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.*” Clearly, this to-and-fro continues 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 a result of a combination of suboptimal plan selection by the database due to incorrect cost models, or lock contention of the database tables, or CPU saturation of the database servers, or congestion on the controller ports, etc. The end-result is lack of end-to-end correlated information causing CIOs to either *throw iron at the problem* and re-create resource silos, or to employ highly paid consultants who understand both databases and SANs to solve the performance problem tickets.

## 1.1 Integrated Diagnosis Challenges

Simplifying the process of root-cause analysis is a complex problem. Enterprise production environments are continuously evolving with configuration changes such as creation of a new volume within a storage controller pool, changes to the switch zoning configuration, addition of new application servers, etc. In addition to resources, the workload characteristics of databases as well as other applications sharing the SAN also vary continuously. In such an environment, the key challenges for diagnosis are enumerated as follows:

- *Event cascading*: The cause and effect of a problem may not be contained within a single layer, but manifested across multiple layers (typically referred to as *event flooding*). Analyzing the impact of an event across multiple layers is a nontrivial problem.
- *Noisy monitoring data*: Monitoring in production environments is configured to minimize the impact on the foreground applications. Typically, the monitoring intervals are large (5 minutes or higher), that lead to inaccuracies (referred to as *noisy* data). The monitoring interval averages out the instantaneous effects of spikes and other bursty behavior, making the data less useful.
- *Complex correlation functions between entities*: An integrated analysis involves a large number of entities namely logical database operators, physical SAN devices, logical volumes and pools, etc. Pure machine learning techniques that aim to find correlations or regression functions in the raw data corpus, which otherwise would have been effective within a single layer with limited parameters, will be ineffective in the integrated scenario.
- *High dimensional search space*: Existing diagnosis tools for some commercial databases [6, 3] use a policy-based approach to create a root-cause taxonomy that is complemented with rules. While this approach has its merits of encoding valuable domain knowledge, it may be grow to be too complex since the search space of an integrated diagnosis has many dimensions.

## 1.2 Contributions

Our vision for this paper is to leverage the existing monitoring tools for databases and SANs in developing an integrated database

and SAN management framework that simplifies administrative tasks requiring an understanding of both databases and SANs, e.g., problem diagnosis, resource provisioning, what-if analysis, and disaster recovery planning. As a concrete instance of the integrated functionality, the paper describes our prototype for an integrated diagnosis tool (referred to as DIADS) that spans across the database and the underlying SAN consisting of end-to-end I/O paths with servers, interconnecting network switches and fabric, and storage controllers. Figure 1 shows an integrated database and SAN taxonomy with various logical (e.g., sort and scan operators) and physical components (e.g., server, switch, storage subsystem). To the best of our knowledge, DIADS is the first diagnosis tool that analyzes both SAN and database events in an integrated fashion. The key contributions of this paper are:

- A novel canonical representation of database query operations combined with physical and logical entities from the SAN environment (referred to as *Annotated Plan Graphs*). The representation captures the information required for end-to-end diagnosis and is created using monitoring data from available database and SAN tools.
- An innovative diagnosis workflow that *drills down* progressively from the level of the query to database plans and to operators, and then uses configuration dependency analysis and symptoms signatures to further drill down to the level of performance metrics and events in components. It then *rolls up* using impact analysis to tie potential root causes back to their impact on the query slowdown. The diagnosis is accomplished using a combination of machine learning and domain knowledge
- An empirical evaluation of DIADS on a real-world testbed with a PostgreSQL database running on an enterprise-class storage controller. We describe (and demo) problem injection scenarios including combinations of events at the database and SAN layers, along with a drill-down into intermediate internal results generated by DIADS.

## 2. RELATED WORK

There has been much prior research for performance diagnosis in databases [6, 10] as well as enterprise storage systems [12, 15]. However, 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. Since the performance problem may lie in any one or a combination of database (DB) and SAN layers, an integrated system like DIADS would be a useful and more efficient approach.

Recent studies that have looked at the interdependence between database and storage systems highlight the importance of such an integrated analysis. [14] describes how an inaccurate storage cost model in database optimizer can significantly impact the choice of query execution plans. [13] proposed an end-to-end database and storage planning technique by characterizing storage I/O workload for a given database workload using an independent combination of database and storage analysis. While being in the same spirit, our work brings a much tighter coupling of database and storage information and capturing their interdependence using a novel annotated plan graph abstraction described later.

A tool like DIADS can also be a good complement to fine-grained database diagnosis and tuning tools like Oracle’s Automatic Database Diagnostic Monitor (ADDM) [6]. ADDM is a SQL profiling tool that is used to identify problems in DB executions plans and provide tuning recommendations. [3] describes a similar Microsoft system for SQL Server. Our work complements this research by providing a non-intrusive and low-overhead mode of analysis that

uses historic performance data to diagnose *changes* in query performance. We discuss this synergy later in Section 6.

There has also been significant work in diagnosing performance problems within the systems research community [16, 9]. Broadly these techniques can be split into two categories – (a) systems using machine learning techniques, and (b) systems using domain knowledge. [16, 2] uses statistical techniques to develop models for a healthy machine and use it to identify *sick* machines. On the other hand, systems like [17, 9, 11] use domain knowledge to create a *symptoms* database that associates performance symptoms with underlying root causes. Such databases are often manually created and require a high level of expertise and resources to maintain.

We believe that for a diagnosis tool to be practically useful, a mix of machine learning and domain knowledge will be required. Pure machine learning techniques can be misled due to spurious correlations in data resulting from noisy data collection or event propagation, where a problem in one component causes another component to be impacted. In DIADS, we counterbalance this effect using suitable domain knowledge like component dependencies, symptoms databases and knowledge of query plan and operator relationships.

Next, we describe Annotated Plan Graphs that capture database and storage component behavior in a single integrated abstraction.

## 3. ANNOTATED PLAN GRAPHS

Suppose a query  $Q$  that a report-generation application issues periodically to the database system shows a slowdown in performance. The root cause of this slowdown may lie in the database layer (execution plan becoming suboptimal due to changes in data properties) or storage layer (increased congestion in the storage pool) or a combination of the two. Diagnosing such a problem requires the ability to understand the behavior of not only the database and storage layers during the execution of the query, but also the interaction between the two layers.

The *Annotated Plan Graph (APG)* abstraction provides this precise ability. At a high level, an APG captures all relevant information related to a complete run of a query plan across the database and SAN layers. This includes database and query level data like operator record counts, SAN level data like storage pool and volume performance metrics, as well as dependency information between the database and SAN components, for example, physical storage and disks that impact a database sort operator.

There are several product offerings (e.g., [7, 8]) in the market that collect and persist monitoring data from IT systems. However, these tools cannot trace the flow of requests across multiple layers of a complex system either because such tracing is impossible (e.g., the subsystems are from multiple vendors) or it is impractical (e.g., the load placed on the production system is high). Thus, no current tool provides a convenient abstraction that captures query behavior in a seamless way across the database and the SAN. Using low-overheard monitoring of historic performance data, an APG fulfills this need.

For the implementation of APGs, DIADS uses the IBM Total-Storage Productivity Center [8] to collect monitoring data from multiple layers of the IT stack including database systems and the complete SAN. The collected data is transformed into a tabular format, and persisted as time-series data in a DB2 database. The data collected at the SAN level 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* (e.g., degradation in volume performance, high workload on storage subsystem).

At the database level, DIADS collects two types of data:

- *Query-level data*: For each execution of plan  $P$ , DIADS collects some low-overhead monitoring data per operator  $O \in P$ . The relevant data includes:  $O$ 's start time, stop time, and *record-counts* (estimated and actual number of records in  $O$ 's output).
- *Database-level data* includes common metrics like the number of cache hits, full-table scans, random I/Os, and locks held.

DIADS uses this data to generate APGs. Figure 1 shows an example of an APG instance for Query 2 from TPC-H. This plan consists of 25 operators, denoted  $O_1$ - $O_{25}$ , with 9 leaf operators. We summarize the novel features of APGs:

- APGs are generated from light-weight monitoring data that is readily available in most production environments.
- APGs are views on the monitoring data that combine what DBAs see—e.g., data on query plans—with what SAN administrators see—data from the numerous SAN components and their inter-connections. More importantly, APGs show each administrator what she typically does not get to see. However, APGs are much more than a juxtaposition of these two pieces of data; as discussed next.
- APGs capture the dependency paths of their constituent components. For example, the dependency path of an operator  $O$  is the set of physical (e.g., CPU, database cache, disk) and logical (e.g., volume, workload) system components whose performance can impact  $O$ 's performance. There are *inner* and *outer* dependency paths. The performance of components in  $O$ 's inner dependency path can affect  $O$ 's performance directly.  $O$ 's outer dependency path consists of components that affect  $O$ '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  $O_{23}$  in Figure 1 includes the server, HBA, FCSwitches, storage subsystem, Pool P2, Volume  $V_2$ , and Disks 5-10. The outer dependency path includes Volumes  $V_3$  and  $V_4$  (because of the shared disks) and other database queries. Section 4 discusses how these dependency paths can be *pruned* using correlation analysis.
- Each component in an APG is *annotated* with appropriate monitoring data collected during the plan's execution. For example, the annotation of an operator  $O$  consists of the performance data collected by DIADS for each component  $C$  in  $O$ 's dependency path; this data is collected in the  $[t_b, t_e]$  time interval where  $t_b$  and  $t_e$  are respectively  $O$ 's (absolute) start and stop times for that execution.

Historic data can be seen as a time series of APGs corresponding to the query  $Q$  of interest. The diagnosis problem is to relate  $Q$ 's slowdown to what change in the rest of the system caused the slowdown.

## 4. DESIGN OF DIADS

When the administrator identifies a query  $Q$  as having experienced a slowdown, DIADS invokes the *diagnosis workflow* shown in Figure 2. This workflow “drills down” progressively from the level of the query to plans and to operators, and then further down to the level of performance metrics and events in components. Finally, an impact analysis is done that “rolls up” to tie potential root causes back to their impact on  $Q$ 's slowdown. As we will show in this section, the workflow applies a combination of statistical machine learning and domain knowledge to the APGs collected for  $Q$ . This novel combination provides built-in checks and balances to deal with the challenges listed in Section 1.

### 4.1 Modules in the Diagnosis Workflow

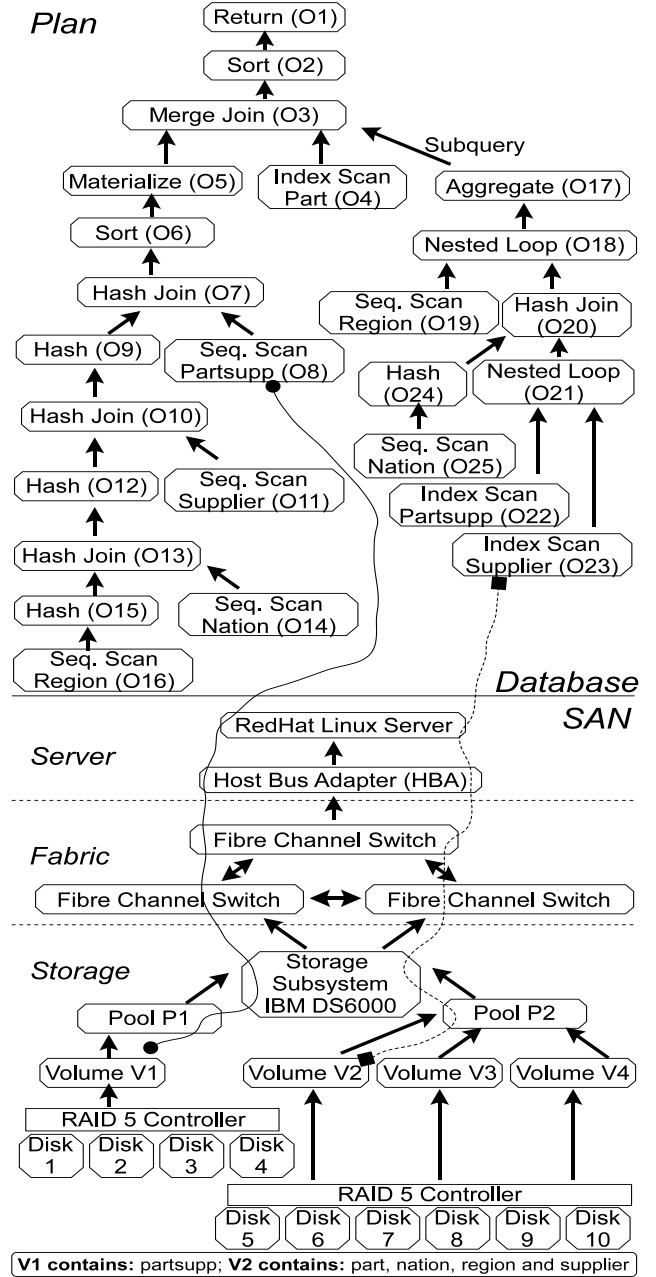


Figure 1: Annotated Plan Graph

The administrator first 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 may be satisfactory, or all runs from 8 AM to 2 PM were satisfactory, and those from 2 PM to 3 PM were unsatisfactory.

**Module Plan Diffing (PD):** The first module in the workflow looks for significant changes between the plans used in satisfactory and unsatisfactory runs. If such changes exist—e.g., if DIADS finds that plan  $P_1$  was used in satisfactory runs and a different plan  $P_2$  was used in unsatisfactory runs—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). Our current implementa-

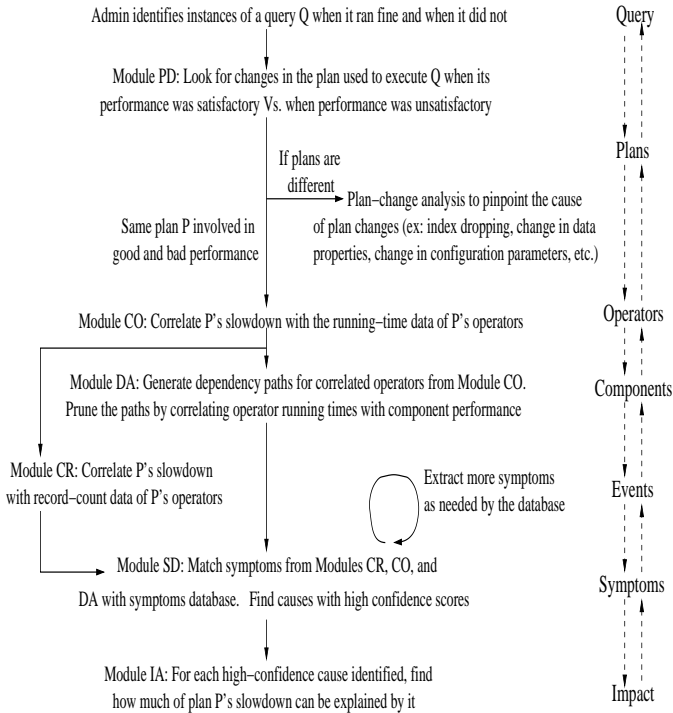


Figure 2: DIADS's diagnosis workflow

tion considers each schema or configuration change that occurred between the runs of  $P_1$  and  $P_2$ , and checks whether this change could have caused the plan change. The remaining modules in the workflow are invoked if DIADS finds a plan  $P$  that is involved in both satisfactory and unsatisfactory runs of the query.

**Module Correlated Operators (CO):** The objective of this module is to find the subset of operators, called the *correlated operator set* ( $COS$ ), whose change in performance best explains plan  $P$ 's slowdown.  $COS$  is identified by analyzing data from satisfactory and unsatisfactory runs of  $P$  which can be seen as records with attributes  $L, t(P), t(O_1), t(O_2), \dots, t(O_n)$  for each run of  $P$ . (Recall the annotations maintained for APGs in Section 3.) 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  $O_i \in P$  for that run. Attribute  $L$  is a *label* representing whether the corresponding run of  $P$  was satisfactory or not.

DIADS feeds this data to *Kernel Density Estimation* ( $KDE$ ) which is a statistical method to estimate the probability density function of a random variable.  $KDE$  applies an estimator to the data to learn the probability density function  $f_i(S_i)$  of the random variable  $S_i$  representing the running time of operator  $O_i$  when  $P$ 's performance is satisfactory. Let  $u$  be an observation of  $O_i$ 's running time when  $P$ 's 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  $O_i$ 's running time when plan performance was unsatisfactory; if so,  $O_i$  belongs to  $COS$ .  $prob(S_i \leq \bar{u})$  is called the *anomaly score* of operator  $O_i$ .

**Module Dependency Analysis (DA):** This module identifies the subset of system components, called the *correlated component set* ( $CCS$ ), such that each component in  $CCS$ : (i) is in the dependency path of at least one operator  $O \in COS$ , and (ii) has at least one performance metric that is significantly correlated with  $O$ 's running time. The fact that a component  $C$  is in the dependency path

of an operator  $O \in COS$  (Property (i) above) does not necessarily mean that  $O$ 's performance has been affected by  $C$ 's performance. Hence, DIADS checks additionally for Property (ii) which is implemented as correlation analysis using  $KDE$ .

**Module Correlated Record-counts (CR):** In this module, DIADS checks whether the change in performance of operators in  $COS$  correlates with their record-counts. Significant correlations mean that data properties have changed between satisfactory and unsatisfactory runs of  $P$ . Once again, correlation analysis is implemented using  $KDE$  to find the *correlated record-count set*  $CRS \subseteq COS$ .

**Module Symptoms Database (SD):**  $COS$ ,  $CCS$ , and  $CRS$  along with other observed SAN and database events may only be *symptoms* of the true root cause of  $P$ 's slowdown. Module  $SD$  seeks to map the observed symptoms to the actual root cause. DIADS generates this mapping using a *symptoms database* whose main purpose is to streamline the use of domain knowledge to (i) create more accurate results by dealing with event propagation, and (ii) generate semantically meaningful results (e.g., reporting lock contention as a cause instead of reporting some performance metrics only). DIADS's implementation of the symptoms database is motivated by an intuitive and commercially-used format called the *Codebook*. The original format assumes a finite set of symptoms such that each distinct root cause  $R$  has a unique *signature* in this set. However, DIADS needs to consider complex symptoms such as symptoms with temporal properties (e.g., contention occurred before failure).

DIADS's symptoms database is a collection of root cause entries each of which has the format  $Cond_1 \& Cond_2 \& \dots \& Cond_z$ , for some  $z > 0$  which can differ across entries. Each  $Cond_i$  is a condition of the form  $\exists symp_j$  (denoting presence of  $symp_j$ ) or  $\neg \exists symp_j$  (denoting absence of  $symp_j$ ). Symptom  $symp_j$  is represented in a high-level language used to express complex symptoms over a base set of symptoms [1]. Each  $Cond_i$  is associated with a weight  $w_i$  such the sum of the weights for each individual root cause entry is 100%. From the symptoms observed currently, DIADS calculates a *confidence score* for each root cause  $R$  as the sum of the weights of  $R$ 's conditions that evaluate to true. We further divide the confidence score into three categories: (i) high (score  $\geq 80\%$ ), (ii) medium ( $80\% > \text{score} \geq 50\%$ ), and (iii) low (score  $< 50\%$ ).

**Module Impact Analysis (IA):** For each high-confidence root cause  $R$  identified by Module  $SD$ , an *impact score* is calculated as the percentage of the query slowdown (time) that can be contributed to  $R$  individually. When multiple problems coexist in the system, impact scores can separate out high-impact causes from the less significant ones. Also, they serve as a safeguard against misdiagnoses resulting from spurious correlations due to noise.

DIADS has multiple implementations of this module. One implementation is an "inverse dependency analysis". First, IA starts from a root cause ( $R$ ) and identifies all system components affected by  $R$ , denoted  $comp(R)$ . The next step is to find the subset of operators ( $op(R)$ ) whose performance is affected by  $comp(R)$ . The impact score is calculated as the percentage of extra running time of  $op(R)$  with respect to the extra plan running time; where extra time is the difference between the average running times across unsatisfactory and satisfactory runs. Another implementation of IA leverages the plan cost models used by database query optimizers.

## 5. EXPERIMENTAL EVALUATION

For the evaluation of DIADS, we considered query slowdowns caused by problems within the database and SAN layers as well as combinations of problems across both layers (a capability which is unique to DIADS). Our experimental testbed is part of a production SAN environment, with the interconnecting fabric and storage

Problem Description	Critical Role of DIADS Modules in Diagnosis
1. SAN misconfiguration leading to contention in volume $V_1$	Identified symptoms pinpoint the correct volume; SD maps symptoms to the correct root cause
2. Contention caused by external workloads on volumes $V_1$ and $V_2$ ; with only the former affecting query performance	DA prunes out the unrelated symptoms and events for volume $V_2$
3. SQL DML causes a subtle change in data properties; problem propagates to SAN causing volume contention	CR identifies the important symptoms; IA rules out volume contention as a root cause
4. Concurrent DB (change in data properties) and SAN (misconfiguration) problems	Both problems identified; IA correctly ranks them
5. DB problem (locking-based) and spurious symptoms of volume contention due to noise	IA identifies volume contention as low impact

**Table 1: Experimental settings of increasing complexity used to evaluate DIADS**

Volume, Perf. Metric	Anomaly Score (no contention in $V_2$ )	Anomaly Score (contention in $V_2$ )
$V_1$ , writeIO	0.894	0.894
$V_1$ , writeTime	0.823	0.823
$V_2$ , writeIO	0.063	0.512
$V_2$ , writeTime	0.479	0.879

**Table 2: Anomaly scores computed during dependency analysis for performance metrics from Volumes  $V_1, V_2$**

controllers being shared by other applications. The testbed runs TPC-H queries on a PostgreSQL database server configured to access tables using two Ext3 filesystem volumes  $V_1$  and  $V_2$  created on an enterprise-class storage controller. Figure 1 shows the table layout and the query plan that we will focus on.

Table 1 gives a high-level summary of our experimental scenarios. DIADS successfully diagnosed the root cause in all these scenarios. Because of space constraints, we give the details of the first scenario only. (The full version of the paper and the demonstration will go through the others.) In this scenario, a contention is created in volume  $V_1$  (from Figure 1) 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  $V_1$ . 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  $V'$ .

**Modules PD and CR:** These two modules correctly identify (respectively) that the plan and the data properties have not changed.

**Module CO:** Based on KDE, this module identifies the set of correlated operators as  $O_2, O_3, O_4, O_6, O_7, O_8, O_{17}, O_{18}, O_{20}, O_{21}$  and  $O_{22}$  (each operator has an anomaly score greater than the threshold of 0.8). This set correctly contains both the leaf operators ( $O_8$  and  $O_{22}$ ) connected to volume  $V_1$ . The eight intermediate operators present in this set are ranked highly because of event propagation: the running times of these operators are affected by the running times of the “upstream” operators (in this case  $O_8$  and  $O_{22}$ ). Finally, operator  $O_4$  is a false positive because it operates on volume  $V_2$ , and is not affected by the contention in  $V_1$ . (As we will see shortly, this false positive caused by noise gets filtered out.)

**Module DA:** This module computes anomaly scores for performance metrics in both volumes  $V_1$  and  $V_2$  since these volumes fall in the dependency paths of the correlated operators. Table 2’s second column shows the anomaly scores for two representative metrics each from  $V_1$  and  $V_2$ . (Table 2’s third column is described later in this section.) As expected, none of  $V_2$ ’s metrics are identified as correlated because  $V_2$  has no contention; while those of  $V_1$  are.

**Module SD:** The symptoms identified so far are: (a) high anomaly

scores for operators using  $V_1$ , (b) high anomaly scores for  $V_1$ ’s performance metrics, and (c) high anomaly score for  $O_4$  (only one out of 7 leaf operators using  $V_2$ ). These symptoms are strong evidence that  $V_1$ ’s performance is a cause of the query slowdown, and  $V_2$ ’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.

Module SD uses a symptoms database that was developed in-house to diagnose query slowdowns.  $V_1$ ’s contention due to the SAN misconfiguration problem was given a high confidence score because all required symptoms are found. (The symptoms database had an entry for this root cause because this problem is very common in production settings.)  $V_1$ ’s contention due to a change in database workload got a medium confidence score because of a weak correlation between the performance of some correlated operators and the rest of the database workload. All other root cause entries in the symptoms database got low confidence scores.

**Module IA:** Impact analysis done using the inverse dependency analysis technique gave 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  $V_1$ .

Next, 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  $V_2$  in a bursty manner such that this extra load had little impact on the query beyond the original impact of  $V_1$ ’s contention. Without intrusive tracing, it would not be possible to rule out the extra load on  $V_2$  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  $V_2$ ’s performance metrics (as shown in the third column in Table 2). However, root causes with contention-related symptoms for  $V_2$  will still have low confidence because most of the leaf operators depending on  $V_2$  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  $V_1$  and  $V_2$ , and attribute both of these as potential root causes. Even worse, the tool may give more importance to  $V_2$  because most of the data is on  $V_2$ . A database-only tool can pinpoint the slowdown in the operators, but it would likely give several false positives like a suboptimal bufferpool setting or a suboptimal choice of execution plan.

Some observations from evaluating DIADS on the broad range of scenarios in Table 1 are:

- Compared to correlation analysis using advanced models (e.g., Bayesian networks [4]), KDE can produce accurate results with

few tens of samples, and is more robust to noise in the data.

- DIADS can deal with (i) database-level problems whose symptoms propagate to the SAN, and vice versa; (ii) independent and concurrent database-level and SAN-level problems; and (iii) spurious and missing symptoms caused by noise.
- DIADS produces good results even when the symptoms database is incomplete. While we expect that entries in the symptoms database are reviewed carefully by administrators, DIADS's own modules like correlation, dependency, and impact analysis can be used to identify important symptoms automatically.

## 6. THE POTENTIAL OF INTEGRATED DATABASE AND SAN TOOLS

While integrated diagnosis using DIADS solves an important practical problem, the proposed system and techniques have the potential to enable even broader functionality. In this section, we present few instances of these capabilities.

- **What-if analysis:** Often database and storage administrators have to apply changes within their respective configurations. In typical enterprises, this process either proceeds without regard to impact on the other layer or requires extensive collaboration between the two teams. In contrast, using techniques developed in our work, it is easy to conceive an integrated database and SAN tool that allows administrators to proactively assess the impact of their planned changes on the other layer. In fact, the impact analysis component of DIADS seems to be a promising approach for developing such a feature. While it may not completely identify all possible problems, it will serve as a valuable check which can then lead to quicker and more focused discussions between the teams.
- **Proactive diagnosis and self-healing:** Another useful extension for DIADS is to provide proactive diagnosis and importantly, self-healing capability. The current symptoms database design can be extended to include, along with symptoms, possible fixes for the root cause of the problem. Once the tool identifies a root cause, it can then apply the fix to self-heal the environment. It is important to note that the fix may be required within the database or storage or a combination of both layers. An integrated approach like ours will be crucial in identifying the right fix and then applying it in any one layer.
- **Integrated Database and SAN Planning:** Along with diagnosis, we believe that annotated plan graphs, by capturing information from the database and SAN layers into a single construct, can lead to smarter planning and optimization for database deployments over a SAN. For example, decisions like the choice of storage required for given database workloads or choice of DB query plan given the storage infrastructure can be intelligently made using these techniques. An early work by Salem et al [13] presented a similar approach for such integrated planning, though it uses a concatenation of independent database and storage analysis components. In contrast, annotated plan graphs provide a much tighter integration with information flow between the two layers aiding in analysis.
- **Machine Learning and Domain Knowledge Interplay:** One of the important aspects of our work is the coupling of machine learning and domain knowledge techniques towards diagnosis. Use of domain knowledge through a symptoms database serves as a guiding tool to the machine learning algorithms preventing spurious correlations due to noisy data or event propagation. An interesting course of future work is to enhance this relationship with machine learning techniques contributing towards identifying potential symptoms which can be checked

by an expert and added to the symptoms database. Considering that a symptoms database may never be complete, this provides a self-evolving mechanism towards bettering the quality of the symptoms databases.

- **Synergy between DIADS and ADDM [6]:** A possible deployment of DIADS is along with a more fine-grained diagnosis tool like Oracle ADDM [6, 5] which uses instrumented code to get operator level timing information. Both use a similar mechanism of finding symptoms and then mapping them to a root cause. However, our use of historic performance data helps in answering questions like *why did my query slow down?* while ADDM helps answering questions like *why is my query slow?*. Combination of both tools provides a stronger analysis engine.

## 7. REFERENCES

- [1] Ana Biazetti and Kim Gajda. *Achieving complex event processing with Active Correlation Technology*. [www-128.ibm.com/developerworks/autonomic/library/ac-acac](http://www-128.ibm.com/developerworks/autonomic/library/ac-acac).
- [2] S. Basu, J. Dunagan, and G. Smith. Why did my pc suddenly slow down? In *Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques (SYSML)*, pages 1–6, 2007.
- [3] S. Chaudhuri, A. C. König, and V. R. Narasayya. Sqlcm: A continuous monitoring framework for relational database engines. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 473–485, 2004.
- [4] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. In *Proc. of the USENIX Operating Systems Design and Implementation (OSDI)*, Dec. 2004.
- [5] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait, and M. Ziauddin. Automatic sql tuning in oracle 10g. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1098–1109, 2004.
- [6] K. Dias, M. Ramacher, U. Shaft, V. Venkataramani, and G. Wood. Automatic performance diagnosis and tuning in oracle. In *Proceedings of Conference on Innovative Data Systems Research (CIDR)*, pages 84–94, 2005.
- [7] Hewlett Packard Systems Insight Manager. <http://h18002.www1.hp.com/products/servers/management/hpsim/index.html>.
- [8] IBM TotalStorage Productivity Center. <http://www-306.ibm.com/software/tivoli/products/totalstorage-data/>.
- [9] A. Manji. Creating symptom databases to service j2ee applications in websphere studio, 2004.
- [10] A. Mehta, C. Gupta, S. Wang, and U. Dayal. Automatic workload management for enterprise data warehouses. *IEEE Data Eng. Bull.*, 31(1):11–19, 2008.
- [11] M. Perazolo. The autonomic computing symptoms format, 2005.
- [12] K. T. Pollack and S. Uttamchandani. Genesis: A scalable self-evolving performance management framework for storage systems. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, page 33, 2006.
- [13] Y. Qin, K. Salem, and A. K. Goel. Towards adaptive costing of database access methods. In *Proc. of Workshop on Self-Managing Database Systems (SMDB)*, 2007.
- [14] F. Reiss and T. Kanungo. A characterization of the sensitivity of query optimization to storage access cost parameters. In *SIGMOD Conference*, pages 385–396, 2003.
- [15] K. Shen, M. Zhong, and C. Li. 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)*, pages 23–23, 2005.
- [16] H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang. Automatic misconfiguration troubleshooting with peerpressure. In *Proceedings of USENIX OSDI Conference*, pages 245–258, 2004.
- [17] S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie. High speed and robust event correlation. *IEEE Communications Magazine*, 34, 1996.

## 8. DEMONSTRATION PROPOSAL

In the demonstration session, we will present DIADS in action on both live faults injected in a realistic DB/SAN environment, as

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

Figure 3: Performance metrics collected by DIADS

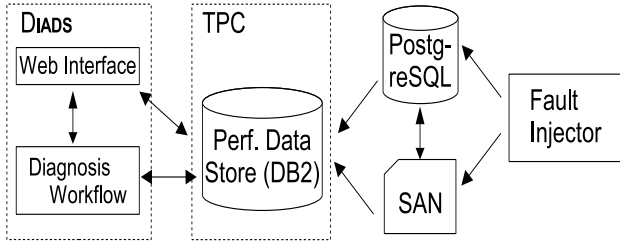


Figure 4: DIADS demonstration setup

well as on offline diagnosis; tracking the reasoning process of the diagnosis workflow by drilling down to the intermediate results like anomaly, confidence, and impact scores. Our demonstration setup is depicted in Figure 4. This setup consists of:

- Data-warehousing queries from the TPC-H benchmark running on a PostgreSQL database server configured to access data on an enterprise-class IBM storage controller.
- The IBM TotalStorage Productivity Center (TPC) running 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 3 shows the key performance metrics collected from the database and SAN. The monitoring data is stored as time-series data in a DB2 database.
- DIADS’s web-based front-end running on a separate server. This front-end supports APG-oriented display and browsing of data collected in the DB2 database. Recall that APGs are views on the monitoring data that combine what DBAs see—e.g., data on query plans—with what SAN administrators see—data from the numerous SAN components and their interconnections. APG-oriented visualization was implemented due to comments from administrators that to diagnose a reasonable fraction of problems all they need to see is the APG diagram.
- DIADS’s diagnosis workflow which is invoked on demand. Each module in the workflow is implemented using a combination of Matlab scripts (for KDE) and Java. DIADS uses a symptoms database that was developed in-house to handle query slowdowns.
- A fault injector that can inject a variety of faults at the database and SAN levels, including SAN misconfiguration, server, disk, or volume contention, RAID rebuilds, changes in data properties, and table-locking problems.

DIADS will be demonstrated through the scenarios listed in Table 1. For the first scenario, we will show live fault injection, namely, a SAN misconfiguration that causes an external workload to use  $V_1$ , causing contention. Because the demonstration period is short (10 minutes), the other scenarios will be demonstrated without live fault injection. Note that DIADS works off of data that is stored

in the DB2 database outside of the system being monitored. This property allows DIADS to run and analyze query slowdowns off-line, provided the required APG data from satisfactory and unsatisfactory runs has already been captured in the DB2 database.

Diagnosis in each scenario starts with the administrator identifying a query that experienced a slowdown, as well as satisfactory and unsatisfactory runs of the query. The diagnosis workflow is then invoked. By default, the workflow is run in a *batch* mode where all modules are executed one after the other, and only the final results are displayed to the administrator. However, DIADS supports an *interactive* mode where results are displayed after each module completes, and the administrator can edit these results before they are fed to the next module. In this mode, the administrator can also re-execute or bypass modules, as well as stop the execution if the desired result is obtained quickly. Our demonstration will use the step-by-step interactive mode to give the audience maximum insight into how DIADS works.

The following features of DIADS will be brought out as we work our way through the five scenarios in Table 1:

- *Dealing with event propagation:* In Scenario #3 in Table 1, we cause a query slowdown by changing the properties of the data, causing extra I/O on Volume  $V_2$ . 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. In this scenario, Module CR correctly identifies all the operators whose record-counts show a correlation with plan performance, causing Module SD to list changes in data properties as a high-confidence cause. Module IA 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.
- *Dealing with multiple concurrent problems:* Scenarios #2 and #4 both have multiple concurrent problems occurring in the system. The difference between them is that in Scenario #2 only one of the problems impacts the query performance, while in Scenario #4 both problems impact query performance (albeit to different levels). DIADS successfully diagnoses the root cause(s) in both scenarios. We will contrast how DIADS works in these two scenarios to showcase the importance of Modules DA and IA.
- *Dealing with noisy monitoring data:* Scenario #5 has one real problem happening in the system. In addition, high levels of noise in the monitoring data cause symptoms corresponding to a different problem to arise by chance. We will show how Module IA enables DIADS to generate the correct diagnosis result even in this challenging scenario.