# DIADS: A Problem Diagnosis Tool for Databases and Storage Area Networks*

Nedyalko Borisov† Shivnath Babu† Sandeep Uttamchandani‡ Ramani Routray‡ Aameek Singh‡
†Duke University ‡IBM Almaden Research Center
{nedyalko, shivnath}@cs.duke.edu {sandeepu, routrayr, aameek.singh}@us.ibm.com

## ABSTRACT

Many enterprise environments have databases running on network-attached storage infrastructure (referred to as *Storage Area Networks* or *SANs*). Both the database and the SAN are complex subsystems that are managed by separate teams of administrators. As often as not, database administrators have limited understanding of SAN configuration and behavior, and limited visibility into the SAN's run-time performance; and vice versa for the SAN administrators. Diagnosing the cause of performance problems is a challenging exercise in these environments. We propose to remedy the situation through a novel tool, called DIADS, for database and SAN problem diagnosis. This demonstration proposal summarizes the technical innovations in DIADS: (i) a powerful abstraction called *Annotated Plan Graphs (APGs)* that ties together the execution path of queries in the database and the SAN using low-overhead monitoring data, and (ii) a diagnosis workflow that combines domain-specific knowledge with machine-learning techniques. The scenarios presented in the demonstration are also described.

## 1. INTRODUCTION

Enterprise business-intelligence applications use databases to store and process terabyte-scale data. Traditionally, storage was attached directly to high-end database servers to meet their capacity, throughput, and bandwidth requirements. However, under-utilization of statically-provisioned hardware and high administration costs for islands of disconnected resources 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 known popularly as a *Storage Area Network (SAN)*.

SANs are very complex systems. A typical SAN has a hierarchy of *core* and *edge* fibre-channel switches with *zoning* configuration that controls the connectivity of server ports with one or more heterogeneous storage controllers. The storage controllers manage a large number of raw disks by aggregating them into logical entities like pools and volumes. Given this complexity, database administrators are forced to treat the SAN as a black-box, entrusting SAN administrators to configure the required CPU, network, and storage resources for meeting the performance requirements of their databases.

While this division of responsibility among database and storage administrators reduces their routine workload, it complicates the task of diagnosing performance problems. In a typical scenario, a database administrator may open a problem ticket for the SAN administrator to analyze and fix issues related to poor performance: *"Queries to the RepDB database used for report generation have a 30% slowdown in response time compared to performance two weeks back."* Unless there is an obvious failure or degradation in the storage hardware or the connectivity fabric, the SAN administrator's response to this problem ticket could be: *"The I/O rate for RepDB tablespace volumes has increased 40%, with increased sequential reads, but the response time is within normal bounds."* This "blame game" may continue for several weeks before the problem is actually fixed. In reality, the query slowdown problem could be due to any number of causes including suboptimal plan selection by the database due to incorrect cost models, lock contention for the database tables, CPU saturation of a database server, congestion in the controller ports, and others.

We are developing an integrated database and SAN management tool, called DIADS[1], that provides administrators with a consistent view of end-to-end system performance as well as the ability to diagnose performance problems automatically. DIADS achieves these goals by: (i) combining the details of both database and SAN operations through a novel abstraction called *Annotated Plan Graphs (APGs)*; and (ii) using a combination of machine-learning techniques and domain knowledge to guide problem diagnosis. Our demonstration will focus on using DIADS to diagnose slowdowns of report-generation queries in databases running over SANs.

## 2. DIADS

This section gives an overview of DIADS. The details can be found in recent DIADS publications [1, 2].

### 2.1 Annotated Plan Graphs

Suppose a query $Q$ that a report-generation application issues periodically to the database system shows a slowdown

*This work is supported by an NSF CAREER award and faculty awards from IBM.


*VLDB '09,* August 24-28, 2009, Lyon, France


[1] *DIA*gnosis for *D*atabases and *S*ANs

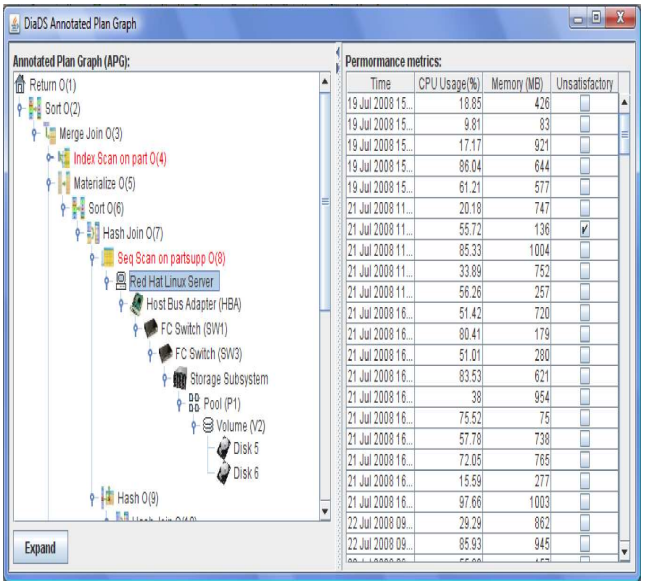


**Figure 1: APG visualization screen in DIADS**

in performance. Diagnosing such a problem requires the ability to understand the behavior of not only the database and storage layers during the execution of $Q$, 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 a comprehensive end-to-end mapping of the database operators in the query plan for $Q$ to the physical disk details where the actual data resides, and everything in between. The screenshot in Figure 1 shows an APG visualized using DIADS's graphical interface.

Some of the novel features of APGs are:

- 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 database administrators see (e.g., data on query plans) with what SAN administrators see (data from numerous SAN components and interconnections). 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.
- 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.
- 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 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.

The APG for a query execution is constructed based on a mix of configuration data and run-time performance data collected from both the database and SAN subsystems. DIADS correlates the database-level data with the SAN-level data to construct the APG.

Data collected at run-time from the database subsystem consists of:

- *Query-level data:* For each execution of plan $P$ for a query, 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 in the plan).
- *Database-level data:* This data includes common metrics like the number of buffer cache hits, full-table scans, random I/Os, and locks held per monitoring interval.

Such data is easily available in most database systems.

The data collected from the SAN subsystem 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 from components, (v) events generated by the system (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). This data is collected using storage management tools like IBM TotalStorage Productivity Center (TPC) [3].

The dependencies among plan operators and SAN components are obtained in the following manner. The mappings of the database *tablespaces* to the storage volumes in the SAN are first extracted from the database configuration file. There are two predominant approaches for associating physical storage to a logical tablespace defined by the database: (a) System Managed Storage (SMS), where the tablespace is mapped to a file system created on a SAN volume; (b) Database Managed Storage (DMS), where the tablespace is created on a raw physical SAN volume with space allocation and associated book-keeping managed directly by the database. All operators in a query plan are related either directly or indirectly to operations on tables; and tables have a one-to-one correspondence to tablespaces. Thus, for each operator $O$, it is possible to map $O$ to the SAN volumes that $O$ depends on. Combining this mapping with the SAN configuration data, we can obtain the inner and outer dependency paths for all plan operators. Each operator is annotated with the performance data of components in its dependency paths.

## 2.2 Diagnosis Workflow

When the administrator identifies a query $Q$ as having experienced a slowdown, DIADS invokes the *diagnosis workflow*. 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 keep DIADS on the right track during diagnosis.

The administrator identifies a query $Q$ as having experienced slowdown by specifying declaratively or marking 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. For example, a change in execution plan for the query could have caused contention in a SAN volume due to the increased I/O in the new plan. 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 change (which includes, e.g., index addition/dropping, changes in data properties, or changes in configuration parameters used in plan selection). Our current implementation pinpoints each schema or configuration change that occurred between the runs of $P_1$ and $P_2$ that 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 the slowdown of plan $P$. For example, the slowdown may be due entirely to a sort operator that exhibits increased running times due to a decrease in memory available to the plan. COS is identified by analyzing data from satisfactory and unsatisfactory runs of $P$ which can be seen as records with attributes $L, t(P), t(O_1), t(O_2), \ldots, t(O_n)$ 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 $O_i \in P$ 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. $Prob(S_i \leq \overline{u})$ is called the *anomaly score* of $O_i$, where $\overline{u}$ is the average of $O_i$'s running time over all samples captured during unsatisfactory performance.

**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. An example of the usefulness of this module is when a scan operator performs poorly due to an unexpected RAID rebuild happening in the SAN volume from which the scan fetches its data. 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 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 $Cond_1$ & $Cond_2$ & ... & $Cond_z$, 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 declarative language used to express complex symptoms. 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% $>$ score $\geq$ 50%), and (iii) low (score $<$ 50%).

**Module Impact Analysis (IA):** For each root cause $R$ identified by Module SD with high confidence, 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. More details are in [1, 2].

# 3. DEMONSTRATION OVERVIEW

Our demonstration will present DIADS in action on problems injected in a realistic DB/SAN environment. The demon-

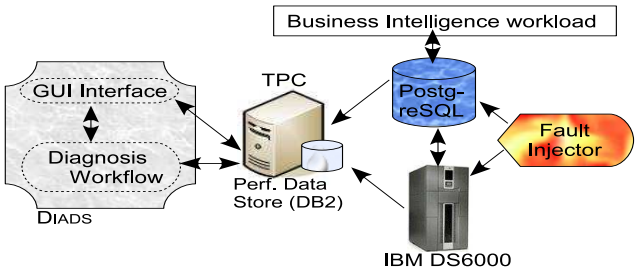


**Figure 2: Demonstration setup**

stration setup, depicted in Figure 2, consists of:

- Data-warehousing queries from the TPC-H benchmark running on a PostgreSQL database server configured to access tables on two Ext3 file-system volumes $V_1$ and $V_2$ created on an enterprise-class IBM storage controller.
- The IBM TotalStorage Productivity Center (TPC) tool 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 TPC). The monitoring data is stored as time-series data in a DB2 database.
- DIADS's graphical 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 database administrators see with what SAN administrators see. APG-oriented visualization was implemented due to feedback from administrators that all they need to see is the APG diagram to diagnose a large fraction of DB/SAN performance problems.
- DIADS's diagnosis workflow which is invoked on demand. Figure 3 shows a screenshot of the workflow. Each module in the workflow is implemented using a combination of R scripts (for KDE) and Java. DIADS uses a symptoms database that was developed in-house to handle query slowdowns.
- A fault injector used for testing and verifying DIADS's results. The injector can cause a variety of faults at both the database and SAN levels, e.g., SAN misconfiguration, server, disk, or volume contention, RAID rebuilds, changes in data properties, and table-locking problems.

DIADS will be demonstrated through several scenarios that keep increasing in diagnosis complexity. (Table 1 in [2] gives our complete list of demonstration scenarios.) Our base scenario consists of a SAN misconfiguration caused by a storage administrator. This misconfiguration places an external workload on volume $V_1$ that affects query performance. We then complicate this base scenario by introducing: (i) problems that propagate through the different components in the system, (ii) concurrent problems, and (iii) noise in the monitoring data. We have designed DIADS to work with monitoring data that is stored in a database outside the system being monitored. This property allows DIADS to run and analyze query slowdowns offline; the required APG data from satisfactory and unsatisfactory runs has already been captured in the monitoring database.

Diagnosis in each scenario starts with the administrator identifying a query that has experienced a slowdown, as well as satisfactory and unsatisfactory runs of that 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—which enables drilling down to intermediate results like anomaly, confidence, and impact scores—to give the audience maximum insight into how DIADS works.

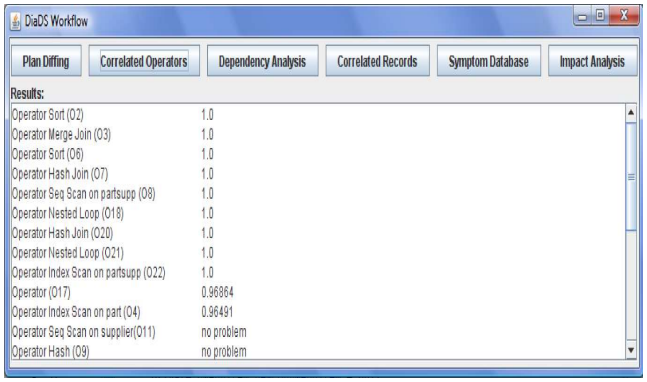


**Figure 3: DIADS's diagnosis workflow being invoked**

Based on viewer interest, we will present how DIADS handles the base scenario along any of the following scenarios:

- *Dealing with event propagation:* A query slowdown is caused through extra I/O on volume $V_2$ by changing data properties in a table. The overall size of all tables remain unchanged. There are no external causes of contention on the volumes. In this scenario, Module CR identifies all the operators whose record-counts are correlated 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:* A bursty workload is introduced on Volume $V_2$ to create multiple concurrent causes of problems. This scenario illustrates the importance of modules DA and IA.
- *Dealing with noisy monitoring data:* Noise is added to the system either by synthetically perturbing the monitoring data or by varying the monitoring interval (which flattens out real spikes in the data, causing correlations to be missed). This scenario illustrates how DIADS performs robustly even in noisy environments.

# 4. REFERENCES

[1] S. Babu, N. Borisov, S. Uttamchandani, R. Routray, and A. Singh. DIADS: Addressing the "My-Problem-or-Yours" Syndrome with Integrated SAN and Database Diagnosis. In *Proc. of Usenix Conf. on File and Storage Technologies (FAST)*, 2009.

[2] N. Borisov, S. Uttamchandani, R. Routray, and A. Singh. Why Did My Query Slow Down? In *Proc. of Fourth Conf. on Innovative Data Systems Research (CIDR)*, 2009.

[3] IBM TotalStorage Productivity Center. `http://www-306.ibm.com/software/tivoli/products/totalstorage-data/`.