

DEMONSTRATION PROPOSAL

AZDBLab: A Laboratory Information System for Large-Scale Empirical DBMS Studies

Young-Kyoon Suh*, Richard T. Snodgrass*, and Rui Zhang⁺

*University of Arizona and ⁺Dataware Ventures

Introduction

Scientific methodology in the database field can provide a deep understanding of DBMS query optimizers, for better engineered designs.

Few *DBMS-centric* labs are available for scientific investigation; prior labs have focused on networks and smartphones.

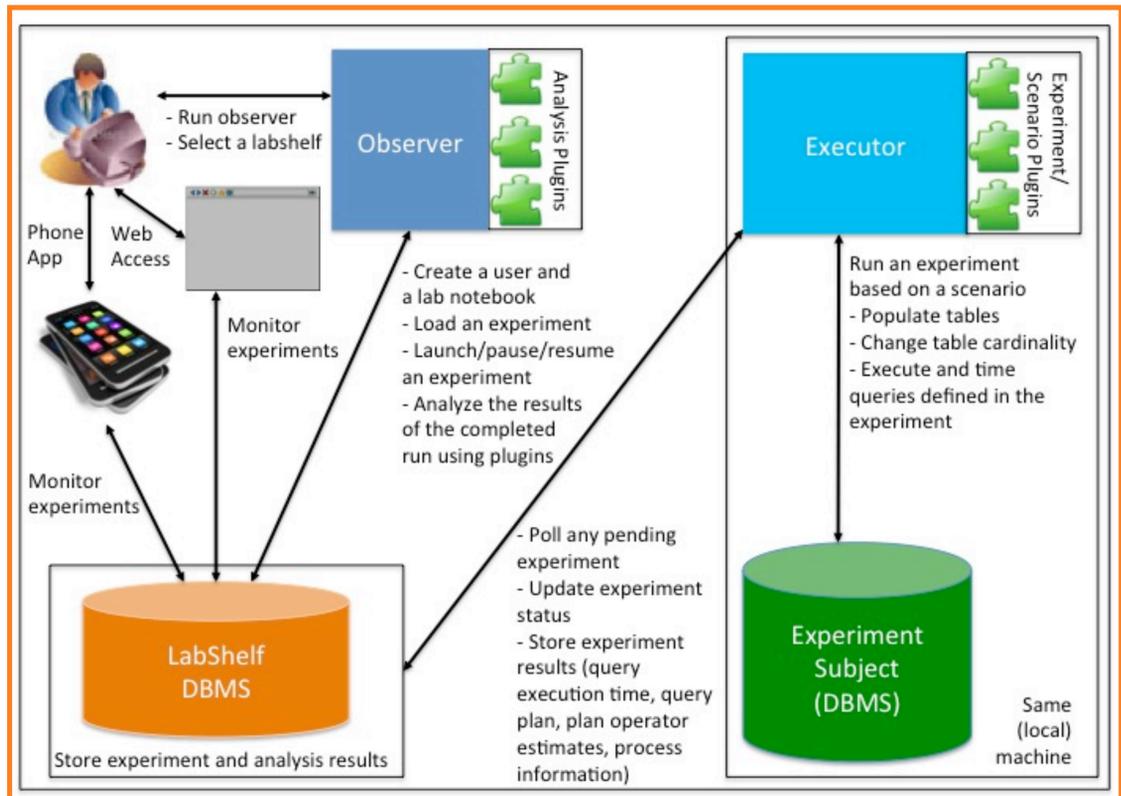
AZDBLAB (AriZona DataBase Laboratory)

- Has been in development for seven years.
- Assists database researchers to conduct *large-scale empirical* studies across *multiple* DBMSes.
- Runs massive experiments with thousands or millions of queries on multiple DBMSes.
- Supports as experiment subjects seven relational DBMSes supporting SQL and JDBC.
- Provides robustness to collect data over **8,277** hours running about **2.4 million** query executions.
- Conducts automated analyses on multiple query execution runs.

Contributions

- Novel research infrastructure, dedicated for large-scale empirical DBMS studies
- Seamless data provenance support
- Several decentralized monitoring schemes: *phone apps*, *web apps*, and *watcher*
- Reusable GUI
- Extensibility through a variety of plugins: labshelf, analysis, experiment subject, and scenario

AZDBLAB Architecture



Demonstration

- Step 1: Choose a labshelf, add a user, and create a notebook, a paper, and a study in the paper on the Observer GUI.
- Step 2: Load an experiment specification into the notebook.
- Step 3: Schedule an experiment run on a particular DBMS.
- Step 4: Monitor the run status via Observer, a web app, and a mobile app, and wait for the experiment to be done.
- Step 5: Add the completed experiment run to the study and conduct a timing protocol analysis for the study.
- Step 6: Produce LaTeX/PDF documents containing the analysis results.

The screenshot shows the Observer GUI. The left pane displays a tree view of the system structure, including Users, LabShelf, Experiment, and Analysis. The right pane shows the 'DBMS Metrology: Measuring Query Time' window. This window contains several tables and a protocol analysis summary.

Description/DBMS	db2	mysql	oracle	pgsql	Total
Cumulative Hours	394.12	721.15	407.00	254.28	1776.55 (hours)

Description/DBMS	db2	mysql	oracle	pgsql	Total
Number of Query Instances	200	200	200	200	800
Number of Q@Cs	2084	555	1253	4848	8740
Number of QEs	20840	5389	12530	48480	87239

Step 1: Performing Pre Sanity Checks
Step 2: Dropping QEs
Step 3: Dropping Q@Cs
Step 4: Calculating Query Time
Step 5: Post Sanity Checks

Protocol Analysis Summary

The measurements were collected using Tucson Timing Protocol Version 2 (TTPv2) on a suite of 4 machines, each an Intel Core i7-870 Lynnfield 2.93GHz quad-core processor on a LGA 1156 95W motherboard with 4GB of DDR3 1333 dual-channel memory and Western Digital Caviar Black TB 7200rpm SATA hard drive, running Red Hat Enterprise Linux Server release 6.4 (Santiago). The protocol provided calculated query evaluation time, including computation and I/O time, in msec. The protocol was utilized exactly as specified. 0 runs violated the experiment-wide sanity checks. Approximately 10.46% of the query executions, 4.03% of the queries-at-cardinality (Q@Cs), and 9.00% of the queries were dropped due to query execution and Q@C sanity checks. As a result, excessive variation in query time was observed in 0.00% of the Q@Cs, 0.94% of the Q@Cs violated strict monotonicity and 0.62% violated relaxed monotonicity. The relative difference of measured and calculated time between retained and dropped Q@Cs was 0.08% and 0.46%, respectively, which is acceptable.

Export LaTeX/PDF