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

Young-Kyoon Suh
University of Arizona
yksuh@cs.arizona.edu

Richard T. Snodgrass
University of Arizona
rts@cs.arizona.edu

Rui Zhang
Dataware Ventures
ruizhang@datawareventures.com

ABSTRACT

In the database field, while very strong mathematical and engineering work has been done, the scientific approach has been much less prominent. The deep understanding of query optimizers obtained through the scientific approach can lead to better engineered designs. Unlike other domains, there have been few *DBMS-dedicated* laboratories, focusing on such scientific investigation.

In this demonstration, we present a novel DBMS-oriented research infrastructure, called *Arizona Database Laboratory* (AZDBLab), to assist database researchers in conducting a *large-scale* empirical study across multiple DBMSes. For them to test their hypotheses on the behavior of query optimizers, AZDBLab can run and monitor a large-scale experiment with thousands (or millions) of queries on different DBMSes. Furthermore, AZDBLab can help users automatically analyze these queries. In the demo, the audience will interact with AZDBLab through the stand-alone application and the mobile app to conduct such a large-scale experiment for a study. The audience will then run a Tucson Timing Protocol analysis on the finished experiment and then see the analysis (data sanity check and timing) results.

1. INTRODUCTION

In the database field, while very strong mathematical and engineering work has been done, the scientific approach has been much less prominent. Much work has focused on proposing new algorithms for optimizing DBMS performance and on building system components for new needs, but the community has not devoted much attention on scientifically understanding DBMS as an *experiment subject.* The deep understanding of query optimizers obtained through the scientific approach can lead to better engineered designs.

There, however, have been few *DBMS-dedicated* laboratories for supporting such scientific investigation, while prior work mainly has focused on networks and smartphones as we will discuss in Section 3.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 40th International Conference on Very Large Data Bases, September 1st - 5th 2014, Hangzhou, China.
Proceedings of the VLDB Endowment, Vol. 7, No. 13
Copyright 2014 VLDB Endowment 2150-8097/14/08.

In this demonstration, we present a novel DBMS-oriented research infrastructure, called *Arizona Database Laboratory* (AZDBLab), to assist database researchers to conduct a *large-scale* empirical study across *multiple* DBMSes.

AZDBLab presents to a scientist an electronic lab notebook. Within this environment, the user can design massive experiments with thousands (or millions) of queries. The user can then begin and monitor many runs of the experiments on different DBMSes over a long period of time.

In addition, AZDBLab provides an integrated and robust database experiment environment. In AZDBLab, a user can record both independent variables (those controlled by the experimenter) and dependent variables (those resulting from the experiment) and perform analyses on the experiment data. For instance, the user can easily combine the completed runs in a study and then analyze in an automated way for the study the query execution (QE) results. In addition, AZDBLab has been sufficiently robust to collect data over 8,277 hours (almost a year) running about 2.4 million query executions.

The design and implementation of AZDBLab has been very challenging. AZDBLab has been developed for seven years by many people, each contributing to different pieces of code. AZDBLab has reached 60K source lines of code. AZDBLab runs on a hardware lab of dedicated machines, one each for each subject DBMS and one to host the DBMS that stores the lab notebooks on a *labshelf.* Having dedicated hardware provides a clear environment for running large-scale, extensive experiments taking days, weeks, or months. A total of seven relational DBMSes (four commercial and three open-source) supporting SQL and JDBC are currently integrated into AZDBLab We will plug more DBMSes into AZDBLab for more solid scientific studies.

The key contributions of this demonstration are as follows.

- We present a novel research infrastructure, AZDBLab, dedicated for a large-scale scientific DBMS study. AZDBLab provides a rigorous abstraction for empirically studying across multiple DBMSes from a variety of perspectives (termed *database ergalics* [2]).
- AZDBLab supports seamless *data provenance* collection within an empirical DBMS study. A database researcher can design and run a substantial experiment with many queries, see the query execution results, perform data sanity check and analysis and make tables, figures, and graphs for the study in an automated, integrated fashion. Note that the data provenance of the study is collected into a labshelf, managed by a central DBMS server.

- For conducting large-scale experiments, AZDBLab provides several *decentralized monitoring* schemes: a stand-alone Java application (named *Observer*), an *Ajax* [1] web app, and a mobile app.

- AZDBLab provides a reusable GUI architecture. Observer's GUI consists of the *tree nodes* in the left-hand side and the *tabbed viewers* in the right-hand side. The tree nodes represent experiment data, while the corresponding tabbed viewers provide the detailed information on the selected data. We have adapted the GUI to several diverse projects.

- AZDBLab provides rich *extensibility*. A variety of *plugins* provides AZDBLab with various analytics and different views of the collected query execution data.

2. MOTIVATION

There are questions concerning fundamental limits of DBMS architectures that simply cannot be answered by investigating a single algorithm or even a single DBMS. Rather, addressing such questions requires the development of *predictive models across multiple DBMSes*.

The objective behind this *scientific* approach is to understand DBMSes as a *general* class of computational artifacts, to come up with insights and ultimately with predictive models about how such systems, again, as a general class, behave by studying multiple DBMSes at a time. These models are articulated and thoroughly tested, in order to understand more deeply the behavior of query optimization and evaluation across multiple DBMSes. They can be eventually used to improve DBMSes through engineering efforts that benefit from the fundamental understanding by this perspective.

AZDBLab has been around over seven years to help us achieve this overarching goal: to *predict important characteristics* of DBMSes to determine *fundamental limits*. AZDBLab allows us to perform substantial experiments (with thousands or more of queries) that quantitatively study these fundamental questions concerning many of the components of a DBMS. These cover (i) *cardinality estimation* (identifying what affects the accuracy of cardinality estimates), (ii) *operator impact* (characterizing how specific types of operators, *e.g.*, join, projection, sorting, affect the accuracy of cardinality estimates, execution time estimates, and optimal plan selection), and (iii) *execution plan search space* (determining its detailed inner structure).

As an experiment tool, AZDBLab coordinates the running, data collection and analysis of such large-scale experiments. It has been broadened to support experiment replication, provenance maintenance, more variety of experiment scenarios, and more DBMSes.

3. RELATED WORK

Although many labs have been established and widely used, their application domains are different from DBMSes. PlanetLab [6] is known as one of the most popular platforms for network research. It allows people to deploy and assess network service [6]. MoteLab [4] was a public testbed for wireless sensor network researchers. Its service was stopped in January 2014.

There are also available mobile computing research labs: CrowdLab [7], SmartLab [5], etc. None of these labs are, however, *DBMS-centric*.

One of the closest tools related to AZDBLab is Picasso [10]. The Picasso tool can also help study query execution plan choices of (three) different DBMSes' cost-based query optimizers, by generating plan and cost diagrams for TPC-H queries. Both Picasso and AZDBLab systems use the "`EXPLAIN`" command, to see the execution plan of a query submitted to DBMSes. Unlike AZDBLab, the Picasso tool neither actually runs queries nor collects the query execution provenance for further analysis.

Using AZDBLab, our empirical DBMS study has yielded two structural causal models; one that identifies across four DBMSes some of the causes of varying query time measures [3] and one that identifies several of the causal factors of suboptimality, when the DBMSes choose a wrong plan. Also, we are currently working on developing another structural causal model to explain DBMS thrashing [11].

4. AZDBLAB SYSTEM OVERVIEW

Figure 1 presents the architecture of AZDBLab. In this section, we describe each component of AZDBLab.

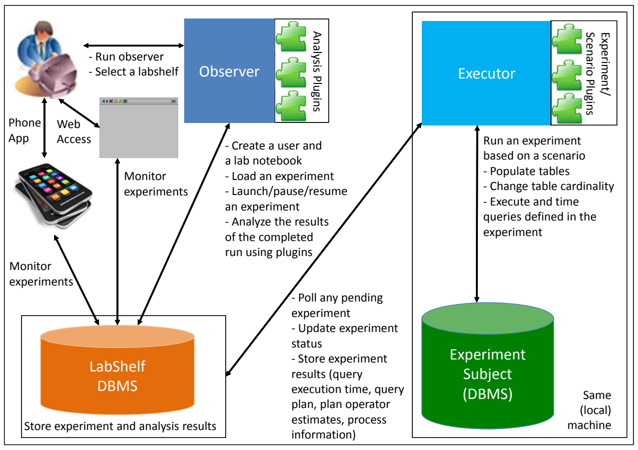

- Run observer
- Select a labshelf
Observer
Analysis Plugins
Phone App
Web Access
Monitor experiments
- Create a user and a lab notebook
- Load an experiment
- Launch/pause/resume an experiment
- Analyze the results of the completed run using plugins
Monitor experiments
LabShelf DBMS
Store experiment and analysis results
- Poll any pending experiment
- Update experiment status
- Store experiment results (query execution time, query plan, plan operator estimates, process information)
Executor
Experiment/ Scenario Plugins
Run an experiment based on a scenario
- Populate tables
- Change table cardinality
- Execute and time queries defined in the experiment
Experiment Subject (DBMS)
Same (local) machine

Figure 1: AZDBLab architecture

4.1 LabShelves

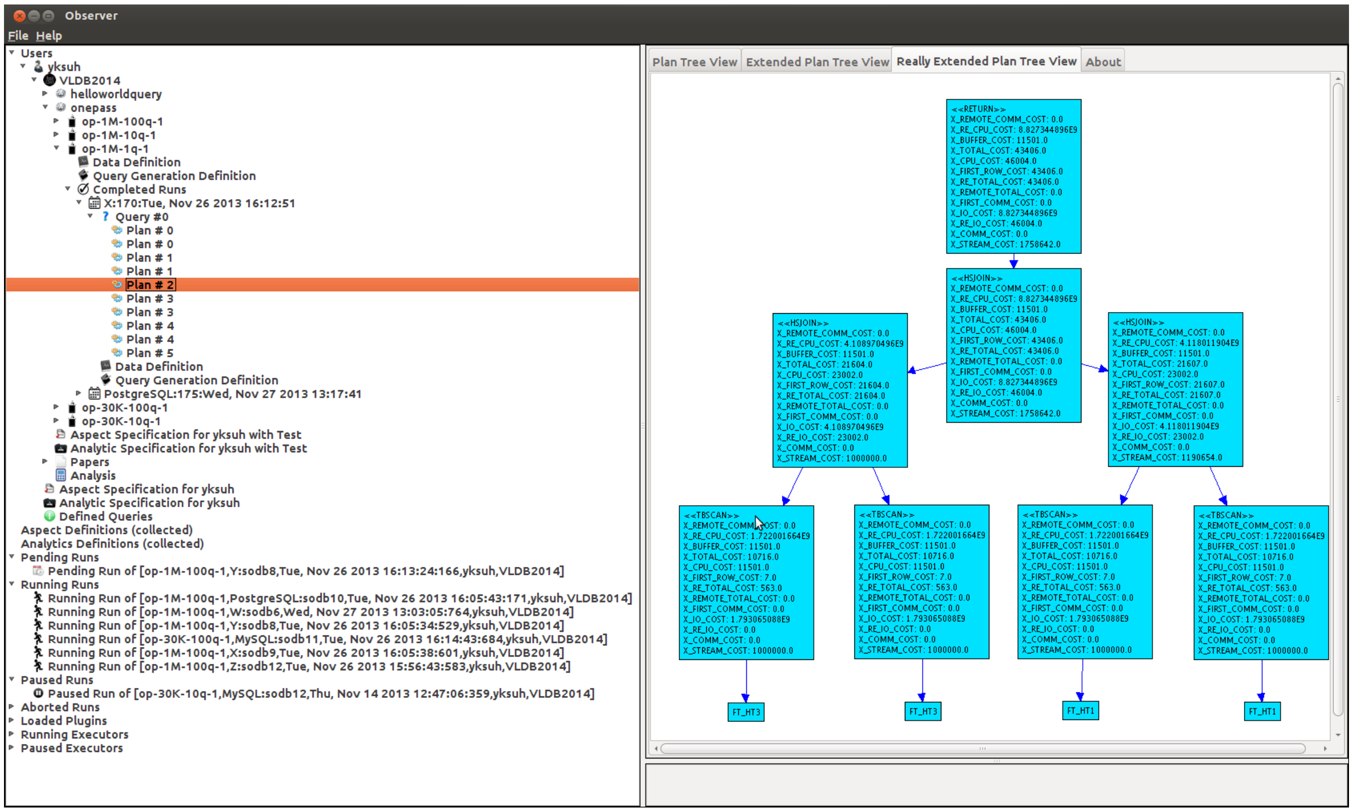
A labshelf comprehensively stores all the *data provenance* related to experiments. It is a fully append-only database [9]. The schema of a labshelf captures *who*, *what*, *when*, *which*, *where*, *why*, and *how*, complying with the *7-W model* [8]. The labshelf schema has been evolving for collecting and analyzing a variety of data relevant to QE. The labshelf data are currently managed by a dedicated DBMS server running on a separate machine.

4.2 Decentralized Monitoring Schemes

In this section, we present a variety of novel, decentralized monitoring schemes being in use in AZDBLab.

4.2.1 Observer

Observer is an integrated GUI for conducting experiments involving a large number of queries and analyzing the results. The GUI is implemented with Java Swing API. When launching Observer, a user can see the main GUI in Figure 2.



```
Users
  yksuh
    VLDB2014
      helloworldquery
      onepass
        op-1M-100q-1
        op-1M-10q-1
        op-1M-1q-1
          Data Definition
          Query Generation Definition
          Completed Runs
            X:170:Tue, Nov 26 2013 16:12:51
              Query #0
                Plan # 0
                Plan # 0
                Plan # 1
                Plan # 1
                Plan # 2
                Plan # 3
                Plan # 3
                Plan # 4
                Plan # 4
                Plan # 5
              Data Definition
              Query Generation Definition
            PostgreSQL:175:Wed, Nov 27 2013 13:17:41
        op-30K-100q-1
        op-30K-10q-1
        Aspect Specification for yksuh with Test
        Analytic Specification for yksuh with Test
      Papers
        Analysis
      Aspect Specification for yksuh
      Analytic Specification for yksuh
      Defined Queries
  Aspect Definitions (collected)
  Analytics Definitions (collected)
Pending Runs
    Pending Run of [op-1M-100q-1,Y:sodb8,Tue, Nov 26 2013 16:13:24:166,yksuh,VLDB2014]
Running Runs
    Running Run of [op-1M-100q-1,PostgreSQL:sodb10,Tue, Nov 26 2013 16:05:43:171,yksuh,VLDB2014]
    Running Run of [op-1M-100q-1,W:sodb6,Wed, Nov 27 2013 13:03:05:764,yksuh,VLDB2014]
    Running Run of [op-1M-100q-1,Y:sodb8,Tue, Nov 26 2013 16:05:34:529,yksuh,VLDB2014]
    Running Run of [op-30K-100q-1,MySQL:sodb11,Tue, Nov 26 2013 16:14:43:684,yksuh,VLDB2014]
    Running Run of [op-1M-100q-1,X:sodb9,Tue, Nov 26 2013 16:05:38:601,yksuh,VLDB2014]
    Running Run of [op-1M-100q-1,Z:sodb12,Tue, Nov 26 2013 15:56:43:583,yksuh,VLDB2014]
Paused Runs
    Paused Run of [op-30K-10q-1,MySQL:sodb12,Thu, Nov 14 2013 12:47:06:359,yksuh,VLDB2014]
Aborted Runs
Loaded Plugins
Running Executors
Paused Executors
```

```
<<RETURN>>
X_REMOTE_COMM_COST: 0.0
X_RE_CPU_COST: 8.827344896E9
X_BUFFER_COST: 11501.0
X_TOTAL_COST: 43406.0
X_CPU_COST: 46004.0
X_FIRST_ROW_COST: 43406.0
X_RE_TOTAL_COST: 43406.0
X_REMOTE_TOTAL_COST: 0.0
X_FIRST_COMM_COST: 0.0
X_IO_COST: 8.827344896E9
X_RE_IO_COST: 46004.0
X_COMM_COST: 0.0
X_STREAM_COST: 1758642.0
```

```
<<HSJOIN>>
X_REMOTE_COMM_COST: 0.0
X_RE_CPU_COST: 8.827344896E9
X_BUFFER_COST: 11501.0
X_TOTAL_COST: 43406.0
X_CPU_COST: 46004.0
X_FIRST_ROW_COST: 43406.0
X_RE_TOTAL_COST: 43406.0
X_REMOTE_TOTAL_COST: 0.0
X_FIRST_COMM_COST: 0.0
X_IO_COST: 8.827344896E9
X_RE_IO_COST: 46004.0
X_COMM_COST: 0.0
X_STREAM_COST: 1758642.0
```

```
<<HSJOIN>>
X_REMOTE_COMM_COST: 0.0
X_RE_CPU_COST: 4.108970496E9
X_BUFFER_COST: 11501.0
X_TOTAL_COST: 21604.0
X_CPU_COST: 23002.0
X_FIRST_ROW_COST: 21604.0
X_RE_TOTAL_COST: 21604.0
X_REMOTE_TOTAL_COST: 0.0
X_FIRST_COMM_COST: 0.0
X_IO_COST: 4.108970496E9
X_RE_IO_COST: 23002.0
X_COMM_COST: 0.0
X_STREAM_COST: 1000000.0
```

```
<<HSJOIN>>
X_REMOTE_COMM_COST: 0.0
X_RE_CPU_COST: 4.118011904E9
X_BUFFER_COST: 11501.0
X_TOTAL_COST: 21607.0
X_CPU_COST: 23002.0
X_FIRST_ROW_COST: 21607.0
X_RE_TOTAL_COST: 21607.0
X_REMOTE_TOTAL_COST: 0.0
X_FIRST_COMM_COST: 0.0
X_IO_COST: 4.118011904E9
X_RE_IO_COST: 23002.0
X_COMM_COST: 0.0
X_STREAM_COST: 1190654.0
```

```
<<TBSCAN>>
X_REMOTE_COMM_COST: 0.0
X_RE_CPU_COST: 1.722001664E9
X_BUFFER_COST: 11501.0
X_TOTAL_COST: 10716.0
X_CPU_COST: 11501.0
X_FIRST_ROW_COST: 7.0
X_RE_TOTAL_COST: 563.0
X_REMOTE_TOTAL_COST: 0.0
X_FIRST_COMM_COST: 0.0
X_IO_COST: 1.793065088E9
X_RE_IO_COST: 0.0
X_COMM_COST: 0.0
X_STREAM_COST: 1000000.0
```

```
<<TBSCAN>>
X_REMOTE_COMM_COST: 0.0
X_RE_CPU_COST: 1.722001664E9
X_BUFFER_COST: 11501.0
X_TOTAL_COST: 10716.0
X_CPU_COST: 11501.0
X_FIRST_ROW_COST: 7.0
X_RE_TOTAL_COST: 563.0
X_REMOTE_TOTAL_COST: 0.0
X_FIRST_COMM_COST: 0.0
X_IO_COST: 1.793065088E9
X_RE_IO_COST: 0.0
X_COMM_COST: 0.0
X_STREAM_COST: 1000000.0
```

```
<<TBSCAN>>
X_REMOTE_COMM_COST: 0.0
X_RE_CPU_COST: 1.722001664E9
X_BUFFER_COST: 11501.0
X_TOTAL_COST: 10716.0
X_CPU_COST: 11501.0
X_FIRST_ROW_COST: 7.0
X_RE_TOTAL_COST: 563.0
X_REMOTE_TOTAL_COST: 0.0
X_FIRST_COMM_COST: 0.0
X_IO_COST: 1.793065088E9
X_RE_IO_COST: 0.0
X_COMM_COST: 0.0
X_STREAM_COST: 1000000.0
```

```
<<TBSCAN>>
X_REMOTE_COMM_COST: 0.0
X_RE_CPU_COST: 1.722001664E9
X_BUFFER_COST: 11501.0
X_TOTAL_COST: 10716.0
X_CPU_COST: 11501.0
X_FIRST_ROW_COST: 7.0
X_RE_TOTAL_COST: 563.0
X_REMOTE_TOTAL_COST: 0.0
X_FIRST_COMM_COST: 0.0
X_IO_COST: 1.793065088E9
X_RE_IO_COST: 0.0
X_COMM_COST: 0.0
X_STREAM_COST: 1000000.0
```

Figure 2: AZDBLab **Observer**

The Observer GUI has a *tree-structured* form. A left-hand tree node is associated with data corresponding to experiment provenance. For instance, '`Users`' node contains users in a labshelf selected by the user. In the figure, under the '`Users`' node, we can see a labshelf user (node), named '`yksuh`'. In the right-hand side, more detailed information on a chosen left-hand node is provided in tabbed views. For the selected plan node ('`Plan # 2`') to be discussed shortly, the corresponding, expanded plan tree is exhibited with various *operator-wise* cost estimates, as shown in Figure 2.

This GUI allows the user to proceed with running a *scenario-based* experiment. A scenario specifies *what an experiment is concerned about* and *what specific steps should be executed*. To run the scenario on AZDBLab, the user first needs to write the corresponding code in Java. For example, a scenario source code, called '`onepass`', can be written to study the query *suboptimality* phenomenon such that when the execution plans of a query change between two adjacent cardinalities (called a *change point*), the actual elapsed time of that query at a lower cardinality is greater than that of a higher cardinality. Once the scenario code is ready, it can be brought as a *plugin* into AZDBLab.

Besides, the user needs to write and load into AZDBLab a simple XML experiment specification, which contains scenario name, schema definition, table population direction, query details, etc. Figure 2 shows several experiments such as '`op-1M-1q-1`' under the '`onepass`' node.

The user can then schedule any of these experiments on a specific DBMS. This scheduled experiment instance is termed as '*experiment run*' and its status is in '*pending*'. Observer updates its GUI to show the pending run.

If an executor, to be discussed in Section 4.3, has any assigned pending run, the executor starts to execute that pending run, whose status then is updated to '*running*'. As the run proceeds, the user can monitor the run's status through this GUI. Note that the user can also *pause* and *resume* the running run if necessary. Once the run is completed, the user can check the experiment results of the run. In Figure 2, two completed onepass runs of DBMS `X` and PostgreSQL DBMS are illustrated. For another study, we have implemented a new scenario and an analysis plugin to observe a totally separate phenomenon, thrashing [11].

4.2.2 Web Apps

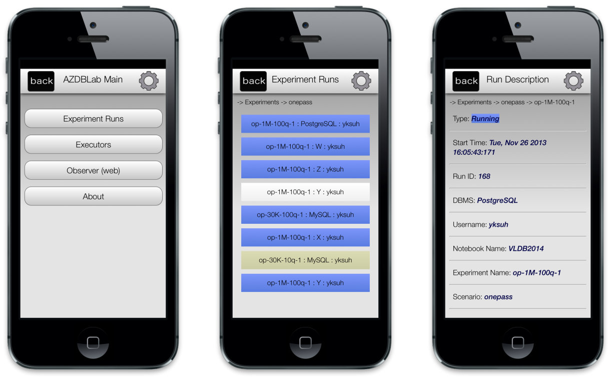
AZDBLab also provides a web app using Ajax [1]. A user can access the web app running on an AZDBLab web server and make a request through a web browser. The web app then invokes an AjaxManager (a Java class), which interacts with the labshelf DBMS server to pull the requested data in a labshelf. The server then responds to the user with the data. The web app provides the same functionalities and GUI as Observer, without requiring direct access to the labshelf server, thereby achieving greater security.

4.2.3 Mobile Apps

To more flexibly monitor the executing runs we also make mobile apps available for the user. We have built the mobile apps on both Android and iOS.

The mobile apps provide simple functionalities, compared to Observer. The user cannot conduct query execution data analysis through the mobile apps, but the user can set up a convenient monitoring environment on the mobile apps.

The mobile apps also make a request to the same AZDBLab web server, which invokes methods from AjaxManager. As mentioned earlier, AjaxManager then connects to the labshelf server and retrieves the data corresponding to the user's request regarding user logins, and experiment and executor statuses.



-> Experiments -> onepass

op-1M-100q-1 : PostgreSQL : yksuh
op-1M-100q-1 : W : yksuh
op-1M-100q-1 : Z : yksuh
op-1M-100q-1 : Y : yksuh
op-30K-100q-1 : MySQL : yksuh
op-1M-100q-1 : X : yksuh
op-30K-10q-1 : MySQL : yksuh
op-1M-100q-1 : Y : yksuh

-> Experiments -> onepass -> op-1M-100q-1

Type: *Running*
Start Time: *Tue, Nov 26 2013 16:05:43:171*
Run ID: *168*
DBMS: *PostgreSQL*
Username: *yksuh*
Notebook Name: *VLDB2014*
Experiment Name: *op-1M-100q-1*
Scenario: *onepass*

(a) Main (b) Exp. Runs (c) A running run

Figure 3: AZDBLab **mobile app**

Figure 3 illustrates the AZDBLab mobile (iOS) app. When a user starts the mobile app, the user is presented with a login screen, into which the user's credentials must be provided. Once the credentials are validated, the user can see the main screen consisting of four menus, as shown in Figure 3(a). '`Experiment Runs`' lists all the runs in a chosen labshelf, as illustrated in Figure 3(b). An item in blue indicates a currently running run, one in white a pending run, and one in gray a paused run. A running run on PostreSQL is exhibited in Figure 3(c). The '`Executors`' menu provides currently running executors, the '`Observer`' menu shows the same view as the web app's one, and the '`About`' menu provides a description of AZDBLab.

4.3 Executor

An executor conducts an experiment on a co-located DBMS, as illustrated in Figure 1. The executor is a stand-alone Java application, utilizing a DBMS experiment subject plugin. A user can launch an executor and schedule a pending run to the executor. The executor can begin the experiment by loading the run's scenario and experiment subject plugins. The executor then creates and populates tables, executes queries, records QE results into AZDBLab, and finishes the run, as explained in Section 4.2.1. If any exception occurs during the experiment, the executor can pause the run. Later, the user can unpause the run. Multiple executors can be in action, perhaps using distinct instances of the same DBMS, each on a different machine. To avoid undesirable latency by network traffic, we ensure that the executor must run on the same machine hosting a DBMS.

5. DEMONSTRATION

Our demo consists of two parts: 1) running experiments with hundreds of queries on different DBMSes and 2) then analyzing QE results from the completed runs.

The goal of the first part of this demo is to show how a database researcher can schedule and run a large-scale experiment in AZDBLab. The steps are as follows.

Step 1: An auditor launches Observer and selects a labshelf. The user then creates a labshelf user and the user's lab notebook. In turn, the user loads into the user's notebook an XML experiment specification guiding table population and referring to queries.

Step 2: The user selects a DBMS, schedules a run of the experiment on the DBMS and then launches an executor for that DBMS. After that, the user can see on console that the experiment gets started on the DBMS. The user may add other DBMSes in order to run the same experiment.

Step 3: The user monitors the run's status via Observer (or the iOS app). At the end, the user will see the run done.

In the second part of this demo, the audience will see how the user can conduct a study on the completed runs.

Step 4: The user creates a *paper* under her notebook and the study under that paper in the Observer GUI.

Step 5: For the study, the user chooses the completed runs via dialog box and executes Tucson Timing Protocol (TTP) on the runs. The automated protocol performs a series of *sanity checks* on QEs of the runs, shows validation results, and calculates query time on the passed QEs.

Step 6: The user finally produces a PDF document containing the protocol analysis results. The user can also create graphs showing the estimated costs of a plan operator (e.g., hash join) over increasing cardinalities. The analysis results and graphs can be used for producing the paper.

6. ACKNOWLEDGEMENT

We thank to T. Buchanan, S. Currim, B. Dicken, M. Johnson, P. Kaslo, A. Kvochko, T. Lowry, H. Zuniga, and others for their contributions to AZDBLab.

7. REFERENCES

[1] W3C, "XMLHttpRequest", January 2014.
[2] R. T. Snodgrass, Database Ergalics, `http://cs.arizona.edu/~rts/ergalics`, viewed Mar 28, 2014.
[3] S. Currim, R. T. Snodgrass, Y-K. Suh, R. Zhang, M. Johnson, and C. Yi, "DBMS Metrology: Measuring Query Time," in *SIGMOD*, pp. 261–272, 2013.
[4] G. Werner-Allen, P. Swieskowski, and M. Welsh, "MoteLab: A Wireless Sensor Network Testbed," in *IPSN*, pp. 483–488, 2005.
[5] G. Larkou, C. Costa, P. G. Andreou, A. Konstantinidis, and D. Zeinalipour-Yazti, "Managing Smartphone Testbeds with SmartLab," in *LISA*, pp. 115–132, 2013.
[6] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir, "Experiences Building PlanetLab," in *OSDI*, pp. 351–366, 2006.
[7] E. Cuervo, P. Gilbert, B. Wu, and O. P. Cox, "Crowdlab: An Architecture for Volunteer Mobile Testbeds," in *COMSNETS*, pp. 1–10, 2011.
[8] S. Ram and J. Liu, "Understanding the Semantics of Data Provenance to Support Active Conceptual Modeling," in *ACML*, pp. 1–12, 2006.
[9] R. T. Snodgrass, **Developing Time-Oriented Database Applications in SQL**, July 1999.
[10] R. Naveen and J. R. Haritsa, "Analyzing Plan Diagrams of Database Query Optimizers," in *VLDB*, pp. 1228–1239, 2005.
[11] A. Thomasian, "Two-Phase Locking Performance and Its Thrashing Behavior," in *ACM TODS*, pp. 579–625, 1993.