# An Automated Approach to Create, Store, and Analyze Large-scale Experimental Data in Clouds

Deepal Jayasinghe, Josh Kimball, Siddharth Choudhary, Tao Zhu, and Calton Pu.
Center for Experimental Research in Computer Systems, Georgia Institute of Technology
266 Ferst Drive, Atlanta, GA 30332-0765, USA.
{deepal, jmkimball, siddharthchoudhary, tao.zhu, calton}@cc.gatech.edu

***Abstract*—The flexibility and scalability of computing clouds make them an attractive application migration target; yet, the cloud remains a black-box for the most part. In particular, their opacity impedes the efficient but necessary testing and tuning prior to moving new applications into the cloud. A natural and presumably unbiased approach to reveal the cloud's complexity is to collect significant performance data by conducting more experimental studies. However, conducting large-scale system experiments is particularly challenging because of the practical difficulties that arise during experimental deployment, configuration, execution and data processing. In this paper we address some of these challenges through Expertus – a flexible automation framework we have developed to create, store and analyze large-scale experimental measurement data. We create performance data by automating the measurement processes for large-scale experimentation, including: the application deployment, configuration, workload execution and data collection processes. We have automated the processing of heterogeneous data as well as the storage of it in a data warehouse, which we have specifically designed for housing measurement data. Finally, we have developed a rich web portal to navigate, statistically analyze and visualize the collected data. Expertus combines template-driven code generation techniques with aspect-oriented programming concepts to generate the necessary resources to fully automate the experiment measurement process. In Expertus, a researcher provides only the high-level description about the experiment, and the framework does everything else. At the end, the researcher can graphically navigate and process the data in the web portal.**

***Keywords*-Automation, Benchmarking, Cloud, Code Generation, Data Warehouse, ETL, Performance, Visualization.**

## I. Introduction

As companies migrate their applications from traditional data centers to private and public cloud infrastructures, they need to ensure that their applications can move safely and smoothly to the cloud. An application that performs one way in the data center may not perform identically in computing clouds [19], so companies need to consider their applications present and future scalability and performance. Neglecting the possible performance impacts due to cloud platform migration could ultimately lead to lower user satisfaction, missed SLAs (Service Level Agreement), and worse, lower operating income. For instance, a study by Amazon reported that an extra delay of just 100ms could result in roughly a 1% loss in sales [27]. Similarly, Google found that a 500ms delay in returning search results could reduce revenues by up to 20% [23].

One of the most reliable approaches to better understand the cloud is to collect more data through experimental studies. By using the experimental measurement data, we can understand *what has happened*, explain *why it happened*, and more importantly *predict what will happen* in the future. Yet, conducting large-scale performance measurement studies introduce many challenges due to the associated complexity of application deployment, configuration, workload execution, monitoring, data collection, data processing, and data storage and analysis. In addition, due to the nature of the experiments, each experiment produces a huge amount of heterogeneous data, exacerbating the already formidable data processing challenge. Heterogeneity comes from the use of various benchmarking applications, software packages, test platform (cloud), logging strategies, monitoring tools and strategies. Moreover, large-scale experiments often result in failures; hence, storing incomplete or faulty data in the database is a waste of resources and may impact the performance of data processing.

We address those challenges through Expertus — a flexible automation framework we have developed to *create*, *store* and *analyze* large-scale experimental measurement data. In Expertus, we have created tools to fully automate the experiment measurement process. Automation removes the error prone and cumbersome involvement of human testers, reduces the burden of configuring and testing distributed applications, and accelerates the process of reliable applications testing. In our approach, a user provides a description of the experiment (i.e., application, cloud, configurations, and workload) through a domain specific language (or through the web portal), and then Expertus generates all of the resources that are necessary to automate the measurement process. Next, the experiment driver uses the generated resources and deploys and configures the application, executes workloads and monitors and collects measurement data.

The main contribution of this paper is the tools and approaches we have developed to automate the data related aspects. To address data processing and parsing challenges, we have used ETL (extract, transform, and load) tools and approaches [21], [22] to build a generic parser (*Expertract*) to process the collected data. The proposed parser can process more than 98% of the most commonly used file formats in our experimental domain. To address the storage challenge, we have designed a special data warehouse called *Experstore* to store performance measurement data. Our data warehouse is fully dynamic that is the tables are created and populated

IEEE IRI 2013, August 14-16, 2013, San Francisco, California, USA
978-1-4799-1050-2/13/$31.00 ©2013 IEEE

on-the-fly based on the experimental data. More specifically, at the end of each experiment, we create a set of tables to store the data, and the schema is solely based on the structure of the data (e.g., how many columns and tables). Finally, to address the challenges associated with navigating and analyzing an enormous amount of performance measurement data, we have built a web portal which helps users to: navigate the data warehouse, visualize the data, statistically analyze the data, and identify interesting performance phenomena from it.

The remainder of this paper is structured as follows. In Section II we provide a high level overview about the experiment automated framework. We discuss our approach to generate heterogeneous measurements data in Section III and in Section IV we discuss about the automated data extractor and the data warehouse we have developed to store measurement data. Section V presents our data analysis tool and we evaluate the effectiveness of our approach in Section VI. Finally, we provide a discussion of related state of the art approaches in Section VII, and we conclude the paper with Section VIII.

## II. Automated Experiment Management Infrastructure

Experiment measurement is a tedious process that consists of multiple activities, and a typical experiment measurement process consists of the following three activities:

- *Create:* preparing the experiment testbed (i.e., cloud) and deploying and configuring the application.
- *Mange:* starting the application components in the correct order, executing workloads, collecting resource monitoring and other performance data, and parsing and uploading the results into the data warehouse.
- *Analyze:* activities associated with analyzing the collected measurement data using various statistical and visualization techniques to understand and explain performance phenomena.

In our approach, we have automated all three activities to provide an efficient way to conduct performance measurement studies. The high level view of our approach, which details these activities and some of the tools used in the process, appears in Figure 1. As shown in the figure, the process consists of eight activities, a brief description of these follows:

- *Experiment Design* is the process of creating a set of experiments that are necessary to evaluate a given application in given target clouds.
- *Expertus* is a code generator which transforms experiment design specifications into deployment, configuration, execution, data collection, and parsing scripts that are essential ingredients to automate the experiment preparation and execution processes.
- *Automation* is the process of using generated scripts to automate platform preparation, application deployment and configuration, experimental execution, data collection, and data processing.
- *Experiments* is the actual execution of workloads and data collection, in fact, most of the Elba publications belong to this category.

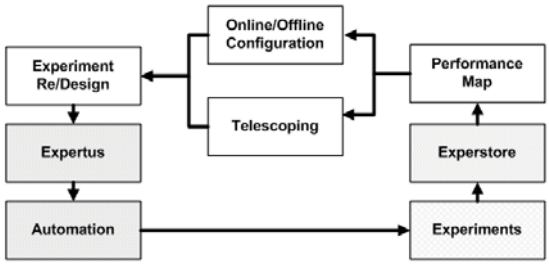


Fig. 1. Our Approach to Large-scale Experiment Measurements

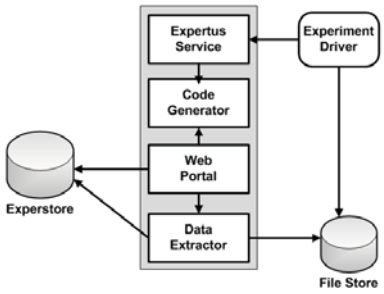


Fig. 2. Key Components of Automated Infrastructure

- *Experstore* is a flexible data warehouse that stores and analyzes resource monitoring and performance data collected through experiment measurements. These data are in fact heterogeneous and vary significantly depending on the experiments and monitoring strategy.
- *Performance Map* is a logical view of experiment results, for example, an application's performance across different clouds.
- *Online/Offline Configuration* is the process of using performance data to make configuration decisions at runtime as well as finding appropriate software settings for new configurations.
- *Telescoping* is the process of using collected data to drive more experiments to deeply understand observed phenomena.
- *Experiment Redesign* is the process of creating new experiments or modifying existing experiments either to validate online/offline configurations or to prove (or disprove) performance hypotheses.

We designed the automation infrastructure by combining multiple modules and built-in flexibility to accommodate new modules. We employed modular architecture because of its distinct advantage of enabling us to change one component without affecting other components. The different modules in the system are illustrated in Figure 2, and a brief description of each is given below:

1) *Code generator:* is the core of the automation which generates all the necessary resources to automate the experiment management process. In a nutshell, code generator takes experiment configuration files as the input and generates all the required files (e.g., scripts).
2) *Expertus Service:* the components of automation framework are connected using SOAP and REST APIs. We

created an Axis2 [**?**] based Web service that supports APIs for code generation, data extraction, status update, and information listing. We used the code generation API to create a command line tool (CMI) for code generation.

3) *Experiment Driver:* We use a centralized approach for experiment execution, and the component called, the experiment driver, is responsible for this task. Code generator generates all the scripts, and a special script called `run.sh`, which maintains the sequence for script execution. Experiment driver uses `run.sh` to find the order of execution. It connects to all the nodes through `SSH/SCP` and executes the scripts on the corresponding nodes. In addition, experiment driver is configured to collect and report information about the user, time, workload start time and end time and the platform to the Expertus service through the REST API.
4) *Data Extraction:* Each experiment produces gigabytes of heterogeneous data for resource monitors (e.g., CPU, Memory, thread pool usage, and etc...), response time and throughput, and application logs. The structure and amount of collected data vary based on system architecture (`64-bits` vs `32-bit`, `2-core` vs. `4-core`), monitoring strategy and monitoring tools (e.g., `sar`, `iostat`, `dstat`, `oprofile`), logging strategy (e.g., Apache access logs), and number of nodes and workloads. Data extractor is written to help users easily export experiment data to the data warehouse.
5) *Filestore:* At the end of each experiment, experiment driver uploads experiment data to a file server to store the data in raw format. Some data analysis tools need to have access to the original data files, so these raw data files need to be retained. In addition, there are temporal files and error logs files, which we do not want to put into the database. Data extractor runs on the file store to export data from the file store into the data warehouse.
6) *Experstore:* Is the flexible, extensible and dynamic data warehouse we have created specifically to store heterogeneous experiment data collected through our experiments.

## III. Automated Generation of Heterogeneous Experimental Data

In our approach to large scale experimental measurements, we deploy actual or representative applications (e.g., benchmarks like RUBBoS [10], RUBiS [11], Cloudstone c [7]) on actual or representative deployment platforms (e.g., Amazon EC2) and execute workloads. Through the large scale experiments, we produce a huge amount of heterogeneous performance data. The heterogeneous nature of the data is arising from the nature of the applications, clouds, monitoring tools, and monitoring strategies. We conduct large-scale experiments and collect data by fully automating the process, and our code generator generates all the necessary resources to automate the process.

The generated resources includes shell scripts as well as other configuration files (e.g., property files, header files, action ordering and etc...). In fact, Expertus is designed to support four key entities in the experiment deployment, configuration and execution. More concretely, it supports *artifacts*, *constraints*, *dependencies*, and *deployment and start order*. These scripts take into account the dependencies among the various system components, including hardware and hypervisor (usually invariant through the experiments), operating system configuration, and server configurations such as the database load on the database server.

Once the system is deployed, we execute the workload against the deployed system. In this step, we run the planned experiments according to the availability of hardware resources. For example, we usually run the experiments by increasing the workload. For each workload, we run the easily scalable (browse only) scenario first, followed by read/write scenarios. To minimize cache inter-dependencies across experiments, after each batch of experiments, we finish the data collection, ramp-down the system, stop all servers, and start the next batch with sufficient ramp-up time. The iterations continue until all the experiments are finished.

During the experiment execution, the automated infrastructure collects information about system resources (e.g., CPU, memory), application specific data (e.g., thread pool usage), application logs (e.g., apache logs), high level data like throughput and response time, and any other data that the user wants to collect. This process continues for each and every workload. In fact, experiments in our domain consist of 50 to 60 workloads, and each workloads runs for approximately 30 minutes. The framework is capable of collecting, managing and storing data without any help from the user. The data extractor can use this data to extract and store in the data warehouse after the experiment is completed.

## IV. Automated Experimental Data Integration and Management

Here we present the data extractor that we have developed to process measurement data and the data warehouse solution that we have developed to flexibly store measurement data.

### A. Expertract - Automated Data Extractor

The large-scale experiment measurements with Expertus generate an enormous amount of metadata in the form of log files, i.e. structured and semi-structured text files. This data needs to be extracted and stored in the environmental data warehouse for later analysis. The primary challenge with this activity is the fact that different tools and software packages produce the data of interest, differently. Thus, the goal of an automated data extractor is to build a generalized parsing approach for experimental data to support both the known and unknown data formats. To begin developing an approach, we explored the more recent and foundational research in the Extract, Transform and Load (ETL) domain. Next, we built a system bound by the existing data files known in the environment, and more specifically, we focused on parsing 'fixed width' flat files.

Generally speaking, the log files that comprise our ETL domain have significant variability. The following categories are just a few of points of difference:

- Structure - semi-structured: [flat files, delimited files, HTML] and structured [XML].
- Data Record Structure - what arrangement or construction within the file represents one data record and how do the data fields relate - either explicitly or implicitly - ontologically.
- Data Type - numeric, string and other ASCII characters.
- Data Variability - how consistent the data is within a specific - either explicit or implicit - data field and across the fields within a file.
- Data Validity - similar to Data Variability but specifically related to identifying error conditions within a specific - either explicit or implicit - data field.

These observations suggest perhaps an alternative view of the original problem. That is any log file contains data and an inherent presentation. This presentation is a mixture of inherent data ontology and human readability factors. Any successful approach must disambiguate these two concerns. Specifically, an approach needs to be able to handle three aspects of any given log file:

- Data - this concerns aspects of data quality and validation.
- Ontology - that is the logical relationship among the data elements in the file.
- Presentation / Layout - this concerns how the data is expressed in the file.

During the course of our system design, we focused on four main monitoring file data patterns, and these cover more than 98% of the monitoring data collected through our experiments. These patterns are:

1) One header: The most basic case, a given file has one header. This header can contain a row describing records and a row describing fields, but it can also just have a row containing fields.
2) Multiple headers with sequentially corresponding data: This pattern is basically (1) except that another header appears later in the file.
3) Multiple headers with non-sequential corresponding data: This pattern differs from (2) in that data later in the file matches the first header in the file at some random position later in the file.
4) Multiple headers appear randomly in the file and data is entirely non-sequential, i.e., randomly distributed throughout the file.

We developed the tool to be user interactive, so a domain expert can help the tool to correctly interpret the data format. At the end of this process, we build an ontology for the file, and then we use the created ontology to process the file during data extraction. We use the previously described approach for unknown or unseen file formats, but we use existing ontologies to describe the file format for the known files. Nevertheless, the matching algorithm that we delivered followed a Greedy

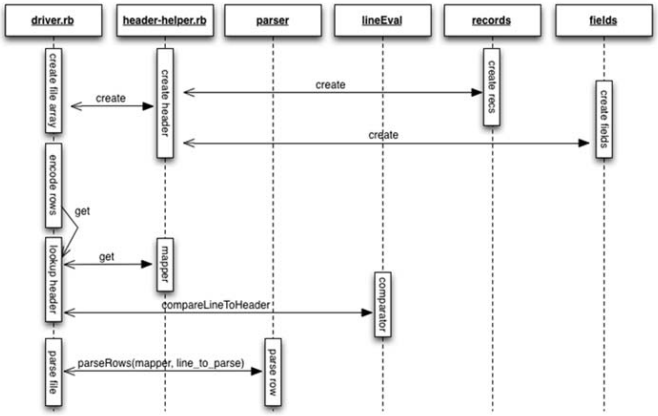


Fig. 3. Experstore - Sequence Diagram for Automated Data Extractor

algorithmic approach–detailed below. We augmented this core functionality with the following surrounding functionality:

- Simple command line user interface to capture user instructions for parsing header rows.
- Object-oriented design that supports separating file contents from file format and layout.
- *Row-Encoding Algorithm*: Only prompt the user when the system thinks the row is a header row. Headers have two distinct characteristics: more alphabetic and special characters relative to the total length of the string. Noise rows, i.e., rows that should be ignored for later processing, have one of two properties but not both, which differentiates them from header rows.
  1) Uses length-weighted character frequency to do first pass encoding.
  2) Then check for the prevalence of special characters.
- *Header-to-Data Row Matching Algorithm*, this algorithm leverages multiple heuristics to achieve its objective:
  - Generate a byte-array representation of the string.
  - Compute character frequencies and scale `weights` based on character frequency. For example, if a `tab` appears once in a string, this character receives significantly more weight than `spaces` that occur in over half of the string. Alphanumeric characters are marked as `0`.
  - If more than one header appears in a document, do a byte-wise comparison of the header row to the row of data of interest. Whichever header-row of data comparison results in the lowest absolute difference is the header selected to process the row.

The sequence diagram shown in Figure 3 represents the primary instruction flow for the parser. It shows the flow of events for: initially loading a data structure to hold the file contents; encoding rows of the data structure to do later matching; and finally, matching rows, classified as a row of data, to the corresponding header.

### *B. Experstore - A Flexible Data Warehouse*

The performance and scalability measurement of enterprise applications is a tedious process, and in most cases, researchers

are unaware of what resources need to be monitored (whether it be high-level data like response time or throughputs or low-level data like resource utilization data and application logs). Moreover, monitoring all the possible resources is not an option, since this might result in enormous performance overhead. Hence, a researcher (or a performance engineer) typically starts with a selected set of resources and gradually changes the monitoring set based on observed results. For example, if the issue is caused by CPU, then a researcher might look into additional data like context switches or interrupts. An experiment can also be deployed on heterogeneous platforms (e.g., XEN vs. KVM, 2-core vs. 4-core), hence the structure of monitoring data may differ from one experiment to another. Frequently, the user changes the monitoring strategy, which results in new data formats, new monitoring data. The user may choose to monitor new resources (possibly using new monitoring tools) based on the observed results. As a result of these issues, the experiments data cannot be feasibly stored on a set of static tables.

In addition, large-scale experiments often result in failures; hence, storing incomplete or faulty data in the database is a waste of resources and may impact the performance of data processing. Most OLAP applications such as experiment data analysis require joining multiple tables or performing self-joins. When the tables are huge, processing becomes very time consuming, and the processing time increases significantly unless the tables can be loaded into the main memory.

We address these problem through Experstore —a special data warehouse to store performance measurement data. Our data warehouse is fully dynamic that is the tables are created and populated on-the-fly based on the experimental data. More specifically, at the end of each experiment we create a set of tables to store the data, and the schema is solely based on the structure of the data (e.g., how many columns, and tables). Tables names are created dynamically by combining experiment ID and timestamp, and names of the tables are stored in a mapping table called 'Resource Mapping Table'. With this approach, if an experiment fails, we can simply drop all the tables. Since we create tables for each experiment, data processing becomes highly efficient, because the small table size (related to each experiment) can be easily joined in memory.

*1) Data to Table Mapping:* As mentioned before, the tables are created dynamically based on the structure of the data. To achieve this, we create an intermediate representation of the data, where for each resource monitoring file we describe how to process the file and which parser to use. One file might contain data for more than one resource type, for example CPU, Memory and Network I/O. For each resource type, we create a 'profile', which maps a file's structure to an applicable schema. As an example, the profile identifies which columns from a CSV file correspond to a database table. Next, we have a mapping, where we specify what profiles are applicable to a given node. A mapping contains node name, file name and corresponding profile. A sample profile and a mapping file is shown below:

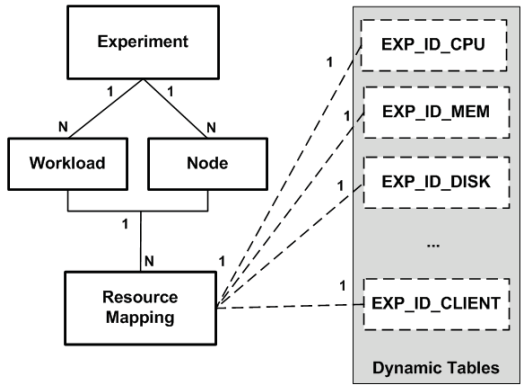


Fig. 4. Experstore - Static and Dynamic Tables

Listing 1. Code Listening for Profile and Mapping

```
<profile>
    <separator>,</separator>
    <resource-name>CPU0</resource-name>
    <processor-class>dataimport.filter.CSVFileProcessor
    </processor-class>
    <column index=''0'' colname=''user'' datatype=''double''/>
    <column index=''1'' colname=''system'' datatype=''double''/>
    <start-index>10</start-index>
    <end-index>0</end-index>
</profile>

<mapping nodename=''Apache'' filname=''169.254.100.3.csv''
          startwith=''false'' endwith=''false''
          profiles=''CPU0,DISK,CPU1,NETWORK,SYSTEM''/>
```

The structure of the data warehouse is shown in Figure 4. As shown in the figure, it consists of four static tables to store experiment metadata (e.g., experiment name, platforms, node and workload information), which are typically fixed across experiments. The highlighted tables are the tables, which are created on-the-fly, as shown in the figure. 'Resource Mapping Table' stores the names of the dynamically created tables along with the resource names. For example, it has a record for CPU utilization for experiment ID (EXP_ID), and the value is EXP_ID_CPU. Likewise, all the monitoring data for a given experiment is stored. In fact, it has a record for each unique node, workload, and resource.

During the course of data loading, a user starts with the results directory and uses the data extraction tools. The tools internally uses Expertract (Section IV-A) to create the ontologies for unknown file formats, and then combine it with known file format to create the mapping file. Finally, the created mapping file is used to export the raw data to the Experstore data warehouse.

## V. Statistical Analysis of Integrated Experimental Data

Experstore is an attempt on our part to aid in the analysis of a large amount of data collected from sundry sources. It equips the user with the capability to identify patterns, trends and relations, which generally gets obscured by the massive quantities of data. Also, it equips the user with capabilities to control the way in which data is represented, which further enhances his productivity and analytical capabilities. The application provides a simple interface that facilitates the comparison of

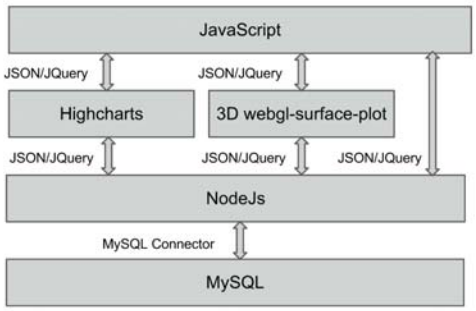


Fig. 6. Architecture - Data Analysis Web Portal

data from seemingly unrelated sources, thereby enhancing the user's ability to see and detect potentially interesting, latent relations that were previously unknown. As an illustration, a examples graph generated is shown in Figure 5.

We believe the application can be really helpful for any system where components have dependencies and hence impact each other. Moreover, the high degree of customization for both graphs and data makes the application highly unique. For example, let's assume we have a process when it is scheduled that it is mostly followed by processes that have heavy disk activity. Such trends are very hard to identify, but our application makes this trivial to pinpoint.

We have tried to keep the design of the application very simple and clean. The core of the application is a `Node.js` server that runs as a webserver. The whole application is written using `Node.js` libraries. On the interface end, the application uses the `HighCharts` framework to plot 2-dimensional graphs and 3D `WebGL-surface-plot` for 3-dimensional graphs. The server and the front-end communicate using jQuery and JSON calls. From the server, we connect to the MySQL database using the Node.js module, which acts as a connector. The high level architecture is shown in Figure 6.

### A. Approach and Features

Throughout the development, we focused on delivering as many customization capabilities in the graphs as possible. Adhering to this principle greatly helped us enhance the utility of the application. Factors such as the variety of graph types, the graphing of different scales, and the ability to vary the values for graph ranges were some of the major challenges that we faced. In order to achieve the desired results, we experimented with different approaches for handling the data.

*1) Fixed vs. Customizable Data Values:* Initially, we designed the application to directly utilize the data that we obtained from the database for graph generation. But later, we realized that this makes comparison between values with significant range differences extremely difficult. Imagine comparing something that varies in thousands to something that has variations in the range of hundreds. We alleviated this condition by allowing mathematical transformations on values before they are plotted on graphs. This allowed us to enhance capabilities. Using the previous example, we could now multiply the values on the second series by a value, 10 in this case, making direct comparison with the first series possible.

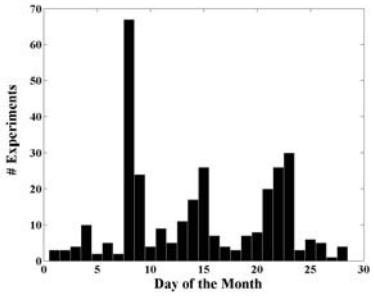


Fig. 7. Number of Experiment Generated During April-2013

*2) Multiple Graphs vs. Multiple Series:* While designing the application, one of the prime concerns was allowing for the comparison among multiple datasets (many series in the same graph?). This needed to be weighed against the amount of information the user would need (how many graphs?). We later took the hybrid approach of allowing the user to have multiple graphs each of which could have multiple series. This greatly enhanced the utility of the application.

*3) Result Migration:* Another prime concern while designing the application was facilitating the sharing of analytical data between users, so collaborative efforts could be fruitful. In order to deliver this feasibly, we had to choose between building printing or export capabilities. The application can currently export graphs as a PNG image, a JPEG image, a PDF document or an SVG vector image.

*4) Advanced Graphs:* To enhance the capabilities of the application, we modularized and integrated advanced features like frequency distribution graphs, bucketed graphs, dynamic graphs, correlation and cumulative graphs and 3-dimensional graphs. This helped us model the application in such a way that future enhancements could be incorporated easily.

## VI. Effectiveness of the Infrastructure

We have used Expertus extensively to perform a large number of experiments on different computing clouds; through experimentation, we have collected a huge amount of data with various data formats, stored them in the data warehouse, and observed interesting performance phenomena.

### A. Active Use of the Tool

Expertus is actively being used for large experimental studies. As an illustration, we have collected data from a number of experiments during April 2013. This is illustrated in Figure 7, and as shown in the figure, we run on average, over 10 parallel experiments daily. In some instances, we run over 50 concurrent experiments.

### B. Amount of Data Collected

Success is considered in the number of different experiments we have performed and amount of data we have collected through those studies. Expertus has been used for years, and it has performed a vast number of experiments: spanning 5 clouds (Emulab [9], EC2 [14], Open Cirrus [12], Wipro [13], and Elba), 3 benchmarks (RUBBoS [10], RUBiS [11], and Cloudstone [7]), 4 database management systems (CJDBC,

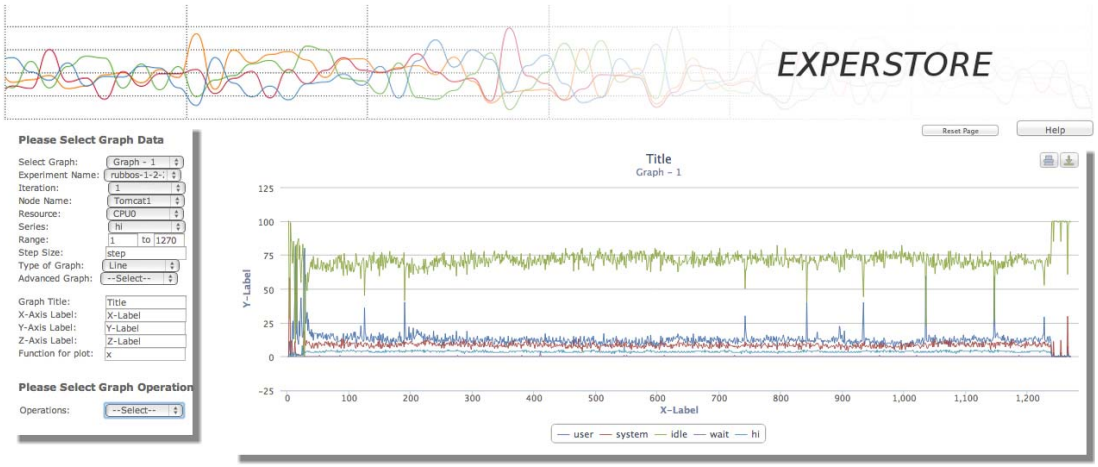


Fig. 5. Data Analysis Web Portal with a Sample Graph

MySQL Cluster, MySQL, PostgreSQL), with various resource monitoring tools (dstat, sar, vmstat), and with different types and number of nodes (i.e., 10 to hundred of servers).

Table I provides a high level summary of the many different experiments performed using RUBBoS, RUBiS, and Cloudstone benchmarks. In the table, experiment refers to a trial of a particular experiment i.e., execution of a particular workload against a combination of hardware and software configurations. Typically, a trial of an experiment takes one hour which is the aggregated value of: reset time, start time, sleeping time, ramp-up time, running time, ramp-down time, stop time, and data copy time. Hence, in Eumlab we have spent approximately 8,000 hours running experiments. In the table, nodes refer to the total number of machines we have used during our experiments. We calculated the number of nodes by multiplying the number of experiments by the number of nodes for each experiment. Configuration means the number of different software and hardware configurations that have been used by our experiments. Finally, the number of data points collected describes the amount of data we collected by running all of our experiments.

TABLE I
NUMBER OF EXPERIMENTS PERFORMED WITH EXPERTUS

| Type | Emulab | EC2 | Open Cirrus | Elba | Wipro |
|---|---|---|---|---|---|
| Experiments | 8124 | 1436 | 430 | 2873 | 120 |
| Nodes | 95682 | 25848 | 4480 | 8734 | 430 |
| Configurations | 342 | 86 | 23 | 139 | 8 |
| Data points | 3,210.6M | 672.2M | 2.3M | 1328.2M | 0.1M |

### C. Testing for Heterogeneous Data Formats

For the purpose of evaluating the robustness of the parser, the following file patterns were tested: 1) one header, 2) multiple header rows with sequentially corresponding data, 3) multiple header rows with non-sequential corresponding data, and 4) multiple header rows appearing randomly in the file, and the data is entirely non-sequential. We used the collected data, which matched these aforementioned patterns, and Table II outlines the observed results. Not only were the file patterns varied but also the structure of the header. Based on the sample, headers contained either one row or two rows. A header with one row, *Only Field Row Header*, only contained data fields. Alternatively, a header with two rows, *Record & Field Row Header* contained a row, which enumerated the data records, and another row, which listed the corresponding data fields for each record. For this latter case, the numbers of records and fields were varied from 1 to the original maximum values: 8 and 16 respectively. If the headers were correctly matched to the applicable row of data, the specified test received a *PASS* grade; otherwise, it received a *FAIL* grade.

TABLE II
EVALUATION SUMMARY OF SUPPORTED FILE FORMATS

| Pattern | Only Field Row Header | Record & Field Row Header |
|---|---|---|
| One header | PASS | PASS |
| Multiple header (sequentially data) | PASS | PASS |
| Multiple header (non-sequential data) | PASS | PASS |
| Multiple header (randomly headers) | N/A | FAIL |

### D. Data Analysis and Performance Phenomena

We have been using the automated infrastructure heavily for data analysis, and through our analysis process we have observed a number of interesting and non-trivial performance phenomena. We have used most of those findings for our publications [18]–[20], [24], [25], and we continue to use the collected data for more publications.

## VII. RELATED WORK

Benchmarking is an essential approach used in both academia and industry to gain an understanding of system behavior, formation and testing of hypotheses, system configuration and tuning, obtaining solution information, and resolving performance bottlenecks. However, there have been relatively few efforts aimed at building software tools for large-scale testing of distributed applications and to reducing complexity of benchmarking [1]–[6]. The ZOO [3] has been designed to support scientific experiments by providing an experiment management languages and supporting automatic experiment execution and data exploration. Zenturio [4] on the other hand

is an experiment management system for parameter studies, performance analysis and software testing for cluster and grid architectures.

Our project parallels several others using XML and XSLT for code generation. For example, the SoftArch/MTE and Argo/MTE teams have also had favorable experiences using XML + XSLT generators to "glue" off-the-shelf applications together [15], [17]. Likewise, XML+XSLT is advocated for code generation tasks in industry as well [16]. One of the closest to our approach is Weevil [8], which is also focus on workload generation and script creation. In fact, later they observed some of the limitation in their approach and proposed four enhancements to explore richer scenarios and to obtain results with greater confidence [5]. To our knowledge, these efforts have not explored the issues of extensibility, flexibility, or modularity that is presented here in this paper.

## VIII. Conclusion

Expertus, our automated experiment management framework, has been developed to minimize human errors and maximize efficiency when evaluating computing infrastructures experimentally. We have used the framework for a large number of experimental studies and through them we have collected a huge amount of data, which we have used for finding interesting performance phenomena. In this paper, we discussed the use of the infrastructure for efficiently creating, storing and analyzing performance measurement data. The code generator generates the necessary resources to fully automate the experiment measurement process, and then using the generated scripts, users can run experimental studies to actually generate the performance data. The automated data processor processes heterogeneous data and stores this data in a flexible data warehouse, built specifically for measurement data. Finally, the visualization tool helps us to easily navigate and perform statistical analysis on the data warehouse to find interesting performance phenomena. We evaluated the proposed automation framework based on its usage, the amount of data it can accommodate, different monitoring and logs formats it supports, and finally, the overall effectiveness of the approach for the scientific community.

Our future work includes, extending the data parser to support additional data formats, extending the data warehouse to use No-SQL databases, and extending the visualization tool to support more customizable graphing capabilities. Google Fusion Tables [26] provides useful APIs and framework for processing big data, our future work also includes utilizing them for processing performance data.

## Acknowledgment

This research has been partially funded by National Science Foundation by IUCRC/FRP (1127904), CISE/CNS (1138666), RAPID (1138666), CISE/CRI (0855180), NetSE (0905493) programs,and gifts, grants, or contracts from DARPA/I2O, Singapore Government, Fujitsu Labs, Wipro Applied Research, and Georgia Tech Foundation through the John P. Imlay, Jr. Chair endowment. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding agencies and companies mentioned above.

## References

[1] Y Ioannidis, M Shivani, G Ponnekanti. ZOO: A Desktop Experiment Management Environment. In *Proceedings of the 22nd VLDB Conference*, Mumbai(Bombay), India, 1996.

[2] K L. Karavanic, B P. Miller. Experiment management support for performance tuning. In *Proceedings of the 1997 ACM/IEEE conference on Supercomputing*, Mumbai(Bombay), India, 1996.

[3] R Prodan, T Fahringer. ZEN: A Directive-based Language for Automatic Experiment Management of Distributed and Parallel Programs. In *ICPP 2002*, Vancouver, Canada.

[4] R Prodan, T Fahringer. ZENTURIO: An Experiment Management System for Cluster and Grid Computing. In *Cluster 2002*.

[5] Y Wang, A Carzaniga, A L. Wolf. Four Enhancements to Automated Distributed System Experimentation Methods. In *ICSE 2008*.

[6] S Babu, N Borisov, S Duan, H Herodotou, V Thummala. Automated Experiment-Driven Management of (Database) Systems. In HotOS 2009, Monte Verita, Switzeland.

[7] A Fox, W Sobel, H Wong, J Nguyen, S Subramanyam, A Sucharitakul, S Patil, D Patterson. Cloudstone: Multi-Platform, Multi-Language Benchmark and Measurement tools for Web 2.0. In CCA 2008.

[8] Y. Wang, M.J. Rutherford, A. Carzaniga, and A. L. Wolf. Automating Experimentation on Distributed Testbeds. In ASE 2005.

[9] Emulab - Network Emulation Testbed. http://www.emulab.net.

[10] RUBBoS: Bulletin board benchmark. http://jmob.objectweb.org/rubbos.html.

[11] RUBiS: Rice University Bidding System. http://rubis.ow2.org/.

[12] Open Cirrus: Open Cloud Computing Research Testbed. https://opencirrus.org/.

[13] Wipro Technologies. www.wipro.com/.

[14] Amazon Elastic Compute Cloud. http://aws.amazon.com.

[15] Cai, Y., Grundy, J., and Hosking, J. Experiences Integrating and Scaling a Performance Test Bed Generator with an Open Source CASE Tool. In *ASE 2004*.

[16] Sarkar, S. Model driven programming using XSLT: an approach to rapid development of domain-specific program generators In *www.XML-JOURNAL.com. August 2002*.

[17] Grundy, J., Cai, Y., and Liu, A. SoftArch/MTE: generating distributed system test-beds from high-level software architecture descriptions. In *ASE 2001*.

[18] Malkowski, S., Hedwig, M., and Pu, C. Experimental evaluation of N-tier systems: Observation and analysis of multi-bottlenecks. In *IISWC 2009*.

[19] Jayasinghe, D., Malkowski, S., Wang, Q., Li, J., Xiong, P., and Pu, C. Variations in performance and scalability when migrating n-tier applications to different clouds. CLOUD 2011.

[20] Wang, Q., Malkowski, S., Jayasinghe, D., Xiong, P., Pu, C., Kanemasa, Y., Kawaba, M., and Harada, L. Impact of soft resource allocation on n-tier application scalability. IPDPS 2011.

[21] Vassiliadis, Panos. A Survey of Extract-Transform-Load Technology. Integrations of Data Warehousing, Data Mining and Database Technologies: Innovative Approaches (2011).

[22] Baumgartner, R., Wolfgang, G., and Gottlob, G.,. Web Data Extraction System. Encyclopedia of Database Systems (2009): 3465-3471.

[23] Kohavi, R., Henne, R.M., Sommerfield, D. Practical guide to controlled experiments on the web: Listen to your customers not to the HiPPO. In ACM KDD 2007.

[24] Malkowski, S., Jayasinghe, D., Hedwig, M., Park, J., Kanemasa, Y., and Pu, C. Empirical analysis of database server scalability using an n-tier benchmark with read-intensive workload. ACM SAC 2010.

[25] Malkowski, S., Kanemasay, Y., Chen, H., Yamamotoz, M., Wang, Q., Jayasinghe,D., Pu,C., and Kawaba, M., Challenges and Opportunities in Consolidation at High Resource Utilization: Non-monotonic Response Time Variations in n-Tier Applications. IEEE Cloud 2012.

[26] Google Inc. Fusion Tables. http://www.google.com/drive/apps.html#fusiontables

[27] G. Linden. Make Your Data Useful, Amazon, November 2006. [Online]. http://home.blarg.net/~glinden/StanfordDataMining.2006-11-29.ppt