International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 1 | ISSN : 2456-3307

Requirements Evocation and Analysis using ETL in Cloud Environments

Dr. K. Purna Chand
Associate Professor, Department of CSE, B V Raju Institute of Technology, Narsapur, Telangana, India

ABSTRACT

Cloud is an efficient service provider due to its flexibility and scalability. It is very popular by providing excellent services like IAAS, SAAS, and PAAS to the users. But still the cloud remains like a black-box for most of the software engineering requirements. It is very difficult to identify and analyze the most important data which is needed to be placed in the cloud. It is a most challenging task because of the practical difficulties that arise during the configuration, execution, deployment and pre-processing of requirements. In this paper we address some of these challenges through a flexible automation framework. We have automated the processing of different users requirements as well as the storage of it in a data warehouse by using ETL.Finally, we have developed a rich web portal to navigate, visualize and analyze the collected requirements.

Keywords : Automation, Cloud, requirements, Data Warehouse, ETL

I. INTRODUCTION

Requirements engineering (RE) is concerned with the identification of the goals to be achieved by the envisioned system, the operationalization of such goals into services and constraints, and the assignment of responsibilities for the resulting requirements to agents such as humans, devices, and software. The processes involved in RE include domain analysis, elicitation, specification, assessment, negotiation, documentation, and evolution. Getting high quality requirements is difficult and critical. Recent surveys have confirmed the growing recognition of RE as an area of utmost importance in software engineering research and practice.

As companies shifting 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, 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 and able to explain why it happened, and more easy to 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 in-complete or faulty data in the database is a waste of resources and may impact the performance of data processing. 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. 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 described an attempt towards automation of use case driven requirements analysis. 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 (Shallow Parser) 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 to store performance measurement data. we created a set of tables to store the data which is gained from users requirements, 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, visualize and analyze the collected requirements.

II. Related Work

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 Platform (i.e., cloud) and deploying and configuring the application.

Mange: starting the application components in the correct order, executing workloads, collecting requirment monitoring and other performance data, and parsing and uploading the results into the data warehouse.

Analyze: activities associated with analyzing the collected requirements data using various statistical and visualization techniques to understand and explain performance phenomena.

In this 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.

Automation is the process of using generated scripts to automate platform preparation, application deployment and configuration, experimental execution, data collection, and data processing.

Experiments are the actual execution of workloads and requirements elicitation

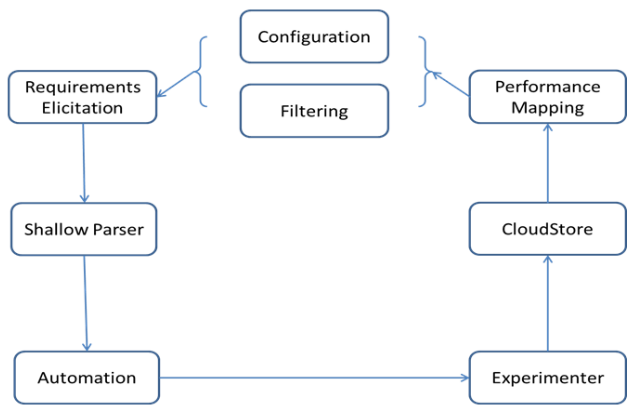

Figure 1 : Automated Framework

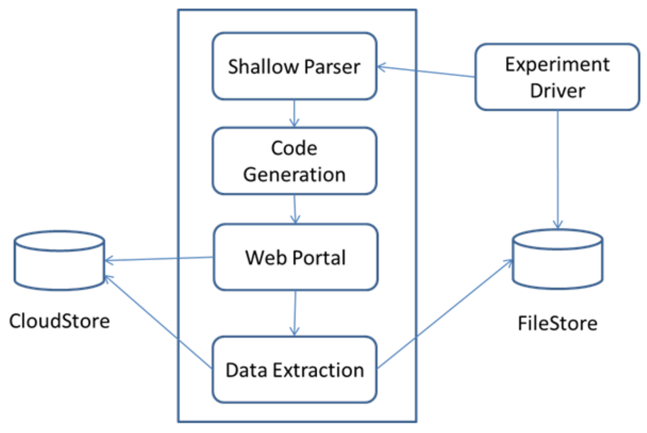

Figure 2 : Different Modules of the Framework

The functionality of every module can be explained as

CloudStore *is a flexible data warehouse that stores and analyzes resource monitoring and performance data col-lected 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 differ-ent clouds.*

Configuration *is the process of using per-formance data to make configuration decisions at runtime as well as finding appropriate software settings for new configurations.*

Tuning *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

Code generator: is the core of the automation which generatesall 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).

Shallow Parser: The components of automation frame-work 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.

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 shallow service through the REST API.

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 work-loads. Data extractor is written to help users easily export experiment data to the data warehouse.

File store: 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.

Cloudstore: Is the flexible, extensible and dynamic data warehouse we have created specifically to store heterogeneous experiment data collected through our experiments.

Requirements Elicitation and Data Extraction and Cloud Storage

In our approach to large scale experimental measurements, we deploy actual or representative applications 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 include shell scripts as well as other configuration files (e.g., property files, header files, action ordering and etc. In fact, shallow parser is designed to the large-scale experiment measurements with S 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 the data fields relate - explicitly or implicitly - ontologically do.
- 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 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

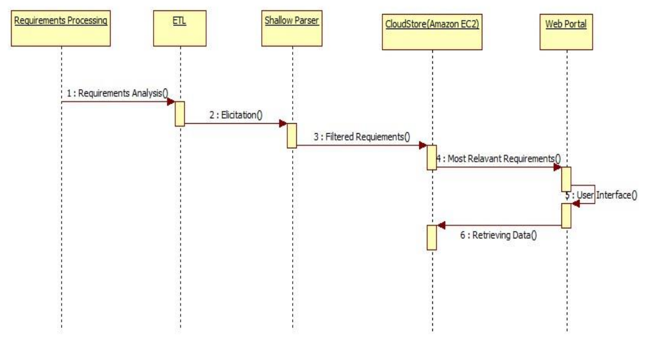

Fig. 3. CloudStore- Sequence Diagram

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 files con-tents from file format and layout.

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.

CloudStore - 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.

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 CloudStore—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.

III. CONCLUSION

Shallow Parser, 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 address some of these challenges through a flexible automation framework. We have automated the processing of different users requirements as well as the storage of it in a data warehouse by using ETL. Finally, we have developed a rich web portal to navigate, visualize and analyze the collected requirements.

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 [23] provides useful APIs and framework for processing big data, our future work also includes utilizing them for processing performance data.

IV. REFERENCES

[1]. Y Ioannidis, M Shivani, G Ponnekanti. ZOO: A Desktop Experiment Management Environment. In Proceedings of the 22nd VLDB Confer-ence, 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]. RUBiS: Rice University Bidding System. http://rubis.ow2.org/.

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

[11]. WIPRO Technologies. www.wipro.com/.

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

[13]. 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.

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

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

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

[17]. 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.

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

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

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

[21]. 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.

[22]. 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.

[23]. 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.