In Proceedings 2015 Computer Measurement Group Conf., San Antonio, TX, November 3-5, 2015.

Automatic Workload Characterization Using System Log Analysis

Mahmoud Awad Daniel A. Menascé
Dept. of Computer Science, MS 4A5
Volgenau School of Engineering
George Mason University
Fairfax, VA 22030, USA
mawad1@gmu.edu menasce@gmu.edu

Abstract

In a previous related work, the authors presented a framework for the dynamic derivation of analytical performance models in autonomic systems. The framework automates the process of deriving and parameterizing Queuing Network (QN) performance models even when detailed knowledge of the system characteristics and user behavior are not readily available. In this paper, we begin to explore and implement the various components of the framework, where we focus on automated workload characterization using system log analysis. In particular, we show an automated technique for generating a Customer Behavior Model Graph (CBMG) by reverse engineering high-level application workflows at the user interface level given the availability of system logs. We ran a number of experiments on the Apache OFBiz ERP system and used Logstash to parse the embedded Apache Tomcat access logs. The results show that our approach for deriving CBMGs is accurate and can be used to estimate the various system workloads.

1 Motivation and Introduction

In a related paper, the authors presented a framework for the dynamic derivation of analytical performance models in autonomic systems [2]. The framework automates the process of deriving and parameterizing Queuing Network (QN) performance models even when detailed knowledge of the system characteristics and user behavior are not readily available. More details in section 4.

In this paper, we begin to explore and implement the various components of the framework, where we focus on automated workload characterization using system log analysis, which is the primary function of the Workload Model Analyzer component. In particular, we show an automated technique for generating a Customer Behavior Model Graph (CBMG) by reverse engineering high-level application scenarios at the user interface level.

Large application systems are characterized by workloads that consist of sessions, where users interact with the system through a sequence of requests. The idea of this paper is to analyze the logs of multi-tier systems so that sessions (sequence of interactions with the system) can be derived and modeled. One of the techniques used to model a repeated sequence of user interactions with application systems is the Customer Behavior Model Graph (CBMG) [8, 9]. CBMG is a state transition diagram used to describe the behavior of a group of users exhibiting similar behavioral patterns while interacting with the application system. The CBMG is also referred to as a User Interaction Model, which describes the nature of interaction between the user and the system. The model describes the frequency of interaction and the sequence of requests submitted to the system [10]

In most cases, the sequence of interactions is dictated by the application itself as it guides the users from one step to the next. The key to accurate characterization of the workload model is to extract the most commonly used patterns, and use those to create an approximate workload model. Outliers can be characterized separately and their impact on the system performance can be analyzed as well. However, the focus of this paper is to find the repeated workflows in user sessions, and we view the outliers as "noise" that, although has an impact on the overall system performance model, it does not represent a significant repeated behavior and, therefore, can be abstracted out of the resulting workload model.

In the related paper, the authors focused on gray box modeling, where they treated the system as a "gray box," where some information about the internal structure of the system is known, such as the number of tiers in a multi-tiered system, along with input and output parameters. In

this paper, however, we assume the availability of system access logs and monitoring logs and we show how to produce a set of CBMGs from these logs. The CBMG is used to perform clustering at the user session level. This clustering is then used to produce the average number of visits to each application module, and consequently, the service demands per application module.

We use the Elasticsearch set of tools generally referred to as the "ELK Stack" (Elasticsearch, Logstash and Kibana). Logstash uses filters to parse log files and store their data in Elasticsearch, which can then be used to search the fields retrieved from the log files for further analysis.

When deriving an analytical performance model for a computer system, it is essential to understand the system architecture (i.e., its hardware and software components and their interconnections), and the behavior of the computer system and its users during operations. For example, how users operate the software (e.g., think time and workload intensity) and how the system responds to normal operations versus operating exceptions. All these aspects contribute to the specifications and parameterization of the performance model.

In this paper, we focus on characterizing the behavior of the system users, the transaction requests they submit and the handling of these requests by the software and hardware components. Input and output server logs, such as database logs, web server logs and various other software-specific logs can be used to characterize the workload in terms of arrival rate, throughput, processing time and the probability of transaction rejection. In particular, we automatically derive the system's CBMGs, which model the flow of transactions from one server to another at the application module level.

The rest of the paper is organized as follows. Section 2 addresses the semantic analysis of system logs and the techniques used to automate this analysis process in order to accurately characterize the workload. Section 3 describes the tools that we used to parse and analyze system logs, produce the CBMGs and the system's workload model. Section 4 describes the CBMG derivation algorithm and our implementation approach. Section 5 shows the results of our experiments using OFBiz, JMeter and the ELK stack, and a comparison between the actual OFBiz scenario and the CBMG produced by our approach. In section 6 we present a few observations and lessons learned from our experiments. Section 7 discusses some related work. Finally, section 8 includes concluding remarks and some of our current efforts related to the automatic derivation of system performance models.

2 Semantic Analysis

We define the semantic analysis of server logs as the process of inferring the relationships between transaction requests for the purpose of improving the accuracy of arrival rates and utilization proportion of the various transaction classes. For example, clustering transactions based solely on response time ignores the implicit relationships between transactions and the trends embedded in the workload.

A big challenge in workload trend analysis is determining user session boundaries and transaction boundaries. A user may submit individual transactions or may submit a series of transactions that follow a prescribed sequence resulting in a workflow that can collectively be categorized as a single transaction. Clustering URL requests in online application systems based on their individual response times means that every request is considered a complete transaction and a complete user session.

Determining session boundaries begins by parsing web server logs using temporal ordering of requests by IP address and request time. In order to differentiate between think time between individual steps in a workflow and the think time between completed sessions, the user behavior has to be analyzed to find behavioral patterns. If multiple users follow the same pattern of requesting

URL1 $\rightarrow$ URL2 $\rightarrow$ URL3 $\rightarrow$ URL4 $\rightarrow$ URL5, for example,

Login $\rightarrow$ Search $\rightarrow$ Browse $\rightarrow$ Add to Cart $\rightarrow$ Purchase, then any long interarrival time between the pattern steps will be considered user think time as opposed to the beginning of a new session.

3 Supporting Tools

The first problem we need to solve is the ability to automatically parse and extract meaningful information from server logs having various formats. Different flavors of web servers, database servers and operating system logs vary in terms of log format between required and optional parameters, their meaning and their data formats. To address this problem, we propose the use of Logstash, which is part of the ELK stack.

ELK stands for Elasticsearch, Logstash and Kibana. Logstash is an open source tool used to collect, organize, parse and store various system logs. Logstash uses templates defined using regular expressions to describe the log and parse its contents. By default, Logstash uses Elasticsearch as its document store, where logs are grouped and stored as indexes. The term index in ElasticSearch is used to refer to the actual data, and it is equivalent to a database in relational database terminology. Relational databases, on the other hand, use the term index to refer to pointers that make it easier to find and access actual data.

Elasticsearch is both a search server and a document store in which documents are stored in JSON format (Java Script Object Notation). Elasticsearch is built on top of Apache Lucene and it provides a native REST API and offers data partitioning, replication, and horizontal scalability by default, and its indexing and data retrieval processes are very fast.

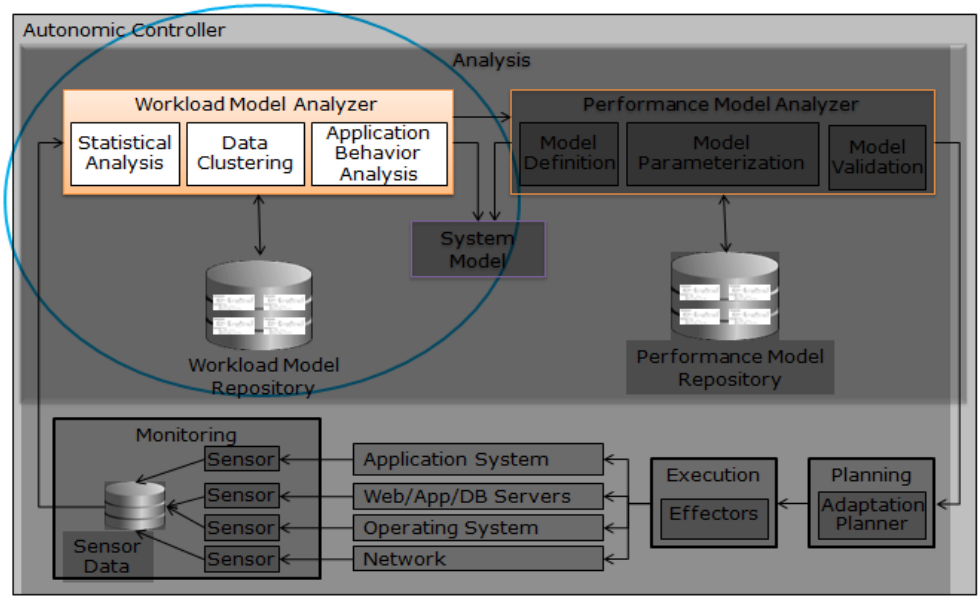

Figure 1: Automatic Model Identification Framework

Kibana is a web-based interface for administering Elasticsearch servers and visualizing its document store contents. Kibana can be customized to provide very elaborate dashboards and is the foundation software for Elasticsearch's Marvel, which is Elastic's commercial tool for monitoring and managing Elasticsearch clusters and nodes.

4 Approach

In [8, 9], the authors presented an algorithm for deriving CBMGs for e-commerce systems by identifying user sessions, which are the main components of these types of systems.

In this section, we show how parts of the automatic derivation and parameterization framework, introduced in [2] and illustrated in figure 1, can benefit from the CBMG derivation algorithm introduced in [8, 9].

The CBMG derivation algorithm identifies the basic components of workloads in e-commerce systems, which are user sessions. The CBMGs -in turn- are used as part of the workload characterization methodology, which includes three main steps; Merge and Filter HTTP logs, Get Sessions and Get CBMGs. In this paper, we build on the authors approach and provide an efficient implementation of their algorithm for Apache webserver access logs. In particular, we show how CBMG diagrams can be used to implement the Application Behavior Analysis component of the Workload Model Analyzer, which is highlighted in figure 1. The Workload Model Analyzer gleans and extracts performance model input parameters, such as arrival rates, various response times and other metrics by parsing server logs and hardware resource utilization logs. The Application Behavior Analysis component, in particular, is used to draw certain conclusions regarding user behavior while interacting with the system.

Next, we describe our implementation of the three main steps of the CBMG-based workload characterization methodology. The high-level approach is illustrated in figure 2:

1. Prepare Logstash configuration file. This is the template used to parse the system log contents by representing the log file contents using regular expression patterns. The results of parsing system log files are stored in Elasticsearch as indexes with the default name of "logstash-%{+YYYY.MM.dd}". In other words, Logstash sorts log file contents by request date and creates an index per day.

   Logstash templates are defined using Grok filters. Basically, Grok filters map the syntactic content of system logs into their semantic representation.

   %{SYNTAX:SEMANTIC}. Example, %{IP:user}

   The other important feature of Grok filters is that they are reusable. Dates and IP addresses, for example, can be defined once and used whenever the system log is expected to have the syntactic equivalent of a date or an IP address.

   For Apache access logs, there are two log formats; the Common Apache Log and the Combined Apache Log. Although the Common Apache Log format is the most commonly used access log format, in our implementation we decided to use the Combined Apache Log since it is the default OFBiz Tomcat access log format. Also, we wanted to incorporate the important attribute "Referrer" and the pattern code %D

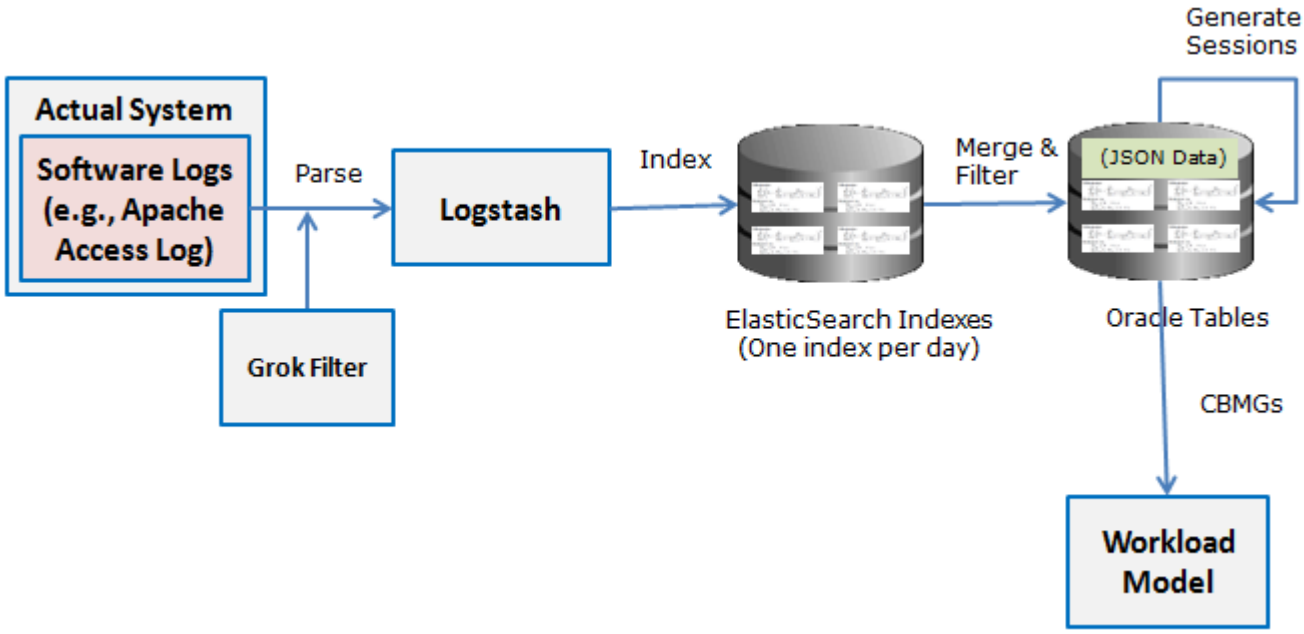

Figure 2: Workload Characterization Using System Log Analysis

(response time in millis) for the future implementations of the framework for the dynamic derivation of analytical performance models.

2. Run Logstash using the configuration file described above (using the command: ../logstash-1.4.2/bin/logstash -f logstash-tomcat.conf). Logstash populated Elasticsearch with five indexes (tables) one per day, where each Apache log entry becomes a document (record) in the corresponding index.

3. Merge and Filter HTTP logs: Because of the implementation complexity of the workload characterization methodology, we decided to load the Elasticsearch indexes into an Oracle database, where we can use the native PL/SQL programming language to develop efficient programs. The Elasticsearch data (documents) from all indexes is loaded into Oracle and stored in a table with a JSON column and cleaned up by removing all special characters and timezones, and -finally- the data is inserted into a new Oracle table in preparation for identifying user sessions.

4. Identify User Sessions: The access log data is now ready to be grouped into sessions. The implementation of this step sets the time threshold between requests (Tau, which indicates a new session) to five minutes. With all the log entries sorted by the user's IP address and request time, Tau is used to determine whether or not the think time between requests is small enough to signify a step in the recently observed user session or the beginning of a brand new user session.

   Note that we had to identify which URL requests should be considered a significant step in the workflow that constitutes a user session. Generating or downloading a PDF file is considered a step in a user session compared to downloading the constituents of a webpage such as JPEG images or JavaScript files. We currently ignore requests to images with the extensions .png, .gif, .jpg and .ico as insignificant in determining the steps that make up a workflow that constitute a complete user session.

5. Get CBMGs: Figure 3 shows a sample CBMG produced by our implementation, where each node shows the unique identifier of the user session step.

5 Experimental Results

In order to test our implementation of the CBMG algorithm, we generated a set of workloads in our testbed. The workloads generated webserver access logs that we used as input to our approach implementation. The objective of this exercise is to compare the actual system workloads with the workloads produced by our approach.

For the testbed, we used Apache JMeter to simulate a number of users accessing Apache OFBiz^{TM} running on an Ubuntu 14.04 server. OFBiz access logs are then parsed using the ELK stack and the resulting parsed access entries are transformed into CBMGs using Oracle database. The hardware used for the testbed consists of three computers; one Ubuntu Linux 14.04 server on which OFBiz software

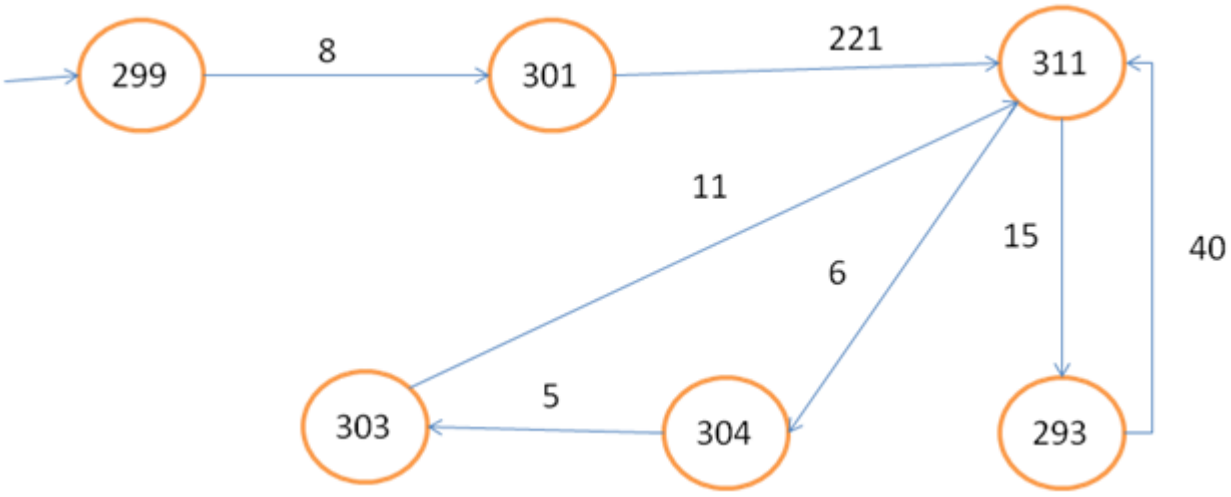

Figure 3: CBMG

was installed in Tomcat 7.0, one Ubuntu Linux 14.04 server on which OFBiz data was loaded into a MySQL database, and a Windows 8 computer hosting JMeter.

Apache OFBiz^{TM} is an open source enterprise automation software from Apache and includes an ERP (Enterprise Resource Planning) application, a CRM (Customer Relationship Management) application, an E-Commerce application and an SCM (Supply Chain Management) application among many other business applications. Apache JMeter was used to generate various workloads. The workloads were chosen to represent various carefully crafted workflows with additional noise represented by random users issuing random individual requests to the OFBiz application system. The added noise is meant to test our algorithm's ability to isolate complete CBMGs from requests that, although may have some impact on the overall system performance, are not part of a use case represented by a complete workflow used by one or more groups of users.

To simulate different groups of users, we used Apache JMeter to run the workflows and the individual isolated user requests. We divided JMeter users into four groups; three running complete workflows from the SCM, accounting and HR applications and the fourth repeatedly running five individual random requests hitting the first page of five OFBiz applications. The resulting Apache access log is used as the only input to our algorithm implementation.

By default, JMeter uses the host computer's IP address to send requests to OFBiz. However, we needed to simulate a number of users with different IP addresses. This is done using "IP spoofing," which is a technique used to send URL requests (i.e., packet requests) using a falsified IP address.

In order to implement IP spoofing in JMeter, we assigned the Windows computer ten static IPv4 addresses, ranging from 192.168.1.130 to 192.168.1.139. This list of addresses is then copied into a .CSV file and referenced in JMeter via a CSV Data Set Config whose Variable Name parameter value is then used in every HTTP request under Source address - IP/Hostname. The IP spoofing configuration is shown in Figures 4 and 5.

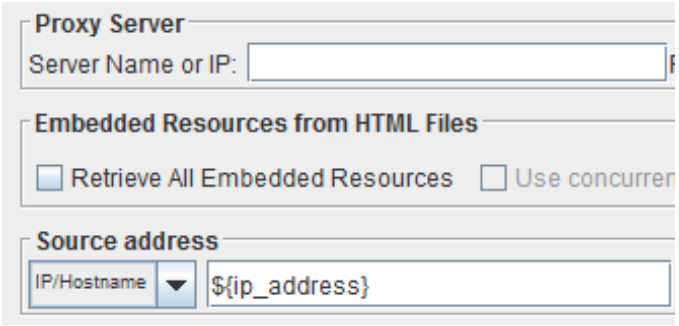

Figure 4: IP Spoofing in JMeter - HTTP Request

Next, we extended the default OFBiz Tomcat access log by adding the pattern code %D in order to capture the time taken to process the request, in milliseconds. The log format is shown in figure 6

The question we posit in our experiments is the following: if we are given access logs of a web-based multi-tier computer system running in production capacity with no additional knowledge provided, could we derive the various CBMGs and -consequently- derive the actual system workloads? As we show below, our approach derived the CBMGs and the system workload. These results are quite encouraging since deriving an approximate workload model is an essential first step in the framework for automatically deriving analytical performance models.

The list below shows the three selected workflows. We ran these worflows in JMeter with 10 users to make full use of the 10 static IP addresses assigned to the host computer

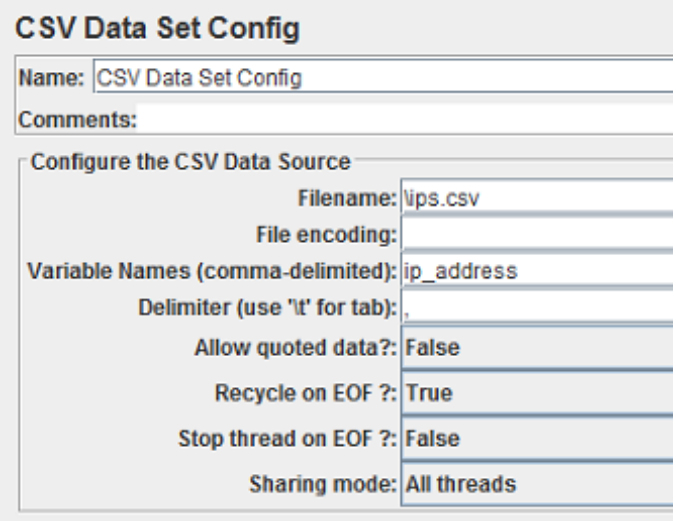

Figure 5: IP Spoofing in JMeter - CSV Data Set Config

```
<property name="access-log-pattern">
  <property-value>
    %h %l %u %t "%r" %s %b "%{Referer}i" "%{User-Agent}i" %D "%S"
  </property-value>
</property>
```

Figure 6: Tomcat Access Log Format

running JMeter.

1. Order Entry: Main Page - Login - Order Entry - Lookup Customer Name - Init Order Entry - Set Order Ship Date - Lookup Product - Add Item (Repeated) - Quick Checkout - Process Order - Order Purchase Report

2. Human Resources: Employees - Lookup Developer1 - Employee Profile - Skills - Add Skill

3. Accounting: Payments - Create New Payment - Create Paycheck for Developer1 - Payments - Find all payments for Developer1

Figures 7 and 8 show the actual steps of the original OFBiz workflow and the derived CBMG, respectively. We define the accuracy of the CBMG derivation by the number of correctly derived steps and their position in the workflow compared to the resulting CBMG. The derived CBMG matches the actual OFBiz workflow except for a few steps in which the think time was less than one second. We explain the reasons in the Discussion section below.

Given the derived CBMG, it is now possible to derive the workload model. The following three graphs 9, 10 and 11 show the number of visits to each state (URL) for sesssion 3 for the user with IP address [192.168.1.230], Total Response Time Per Workflow Per Session, and Average Response Time Per Workflow Across Sessions, respectively. These results are only a sample of the workload model characteristics inferred from the CBMGs generated for the various OFBiz users in our experiments.

6 Discussion

The following are some observations from our experiments and results:

1. As described before, we define Tau to be the time threshold between requests that would signify that the following request is the first request in a new session for the same customer and not just think time between requests of the same session. During our experiments, we observed that the value of Tau can be somewhat subjective. In some cases, the users are fully accustomed to performing a number of tasks rapidly without a significant delay or think time in between. A user of Apache OFBiz may enter a product order, enter a new payment and modify the skills of an employee in the human resources database with little think time in between.

   One might argue that a user session may include all three tasks. However, this assumption would make it hard to perform semantic analysis for the purpose of grouping tasks that belong to a complete workflow. This is important in implementing the dynamic performance model derivation framework because accessing the human resource database, for example, may have a much larger resource demand than ordering a product. Lumping the two tasks together will impact the outcome of the framework, i.e., the resulting QN models, since transaction classes in QN models are supposed to describe groups of transactions with similar characteristics. In this case, the model's workload model would end up combining two transaction classes into one.

2. The default time format for logging when a request is issued in Apache webserver and Tomcat access logs is down to the second. Although it is possible to capture request time in milliseconds, the majority of webserver logs will maintain one second accuracy and that is probably acceptable for most monitoring functions. Considering that some HTTP requests may take less than one second, relying solely on the request time to model CBMGs may not yield an accurate sequence of workflow steps. For example, in figure 7, the normal order entry process is to /checkout then /processorder. However, the derived CBMG in figure 8 shows that the user performs a /processorder then a /checkout. Had the request time been in milliseconds, the derived CBMG would have shown the correct order of steps.

3. In the HTTP protocol, using the POST method to submit form requests to the webserver hides the URL parameters. There are cases where the same module can be used to retrieve data from multiple sources

Actual OFBiz Scenario for IP: [192.168.1.230] and Session_id: [3]
URL path == ordermgr/control
/path/main
/path/login
/path/orderentry
/path/LookupUserLoginAndPartyDetails
/path/LookupCustomerName?ajaxLookup=Y& _LAST_VIEW_NAME_=main&searchValueFieldName=partyId
/path/LookupCustomerName
/path/initorderentry
/path/setOrderCurrencyAgreementShipDates
/path/LookupProduct?ajaxLookup=Y& _LAST_VIEW_NAME_=main&searchValueFieldName=add_product_id
/path/LookupProduct
/path/additem
/images/products/GZ-1001/small.png
/images/products/GZ-9290/small.png
/path/quickcheckout
/path/updateCheckoutOptions/quickcheckout
/path/checkout
/path/processorder
/path/getAssociatedStateList
/path/OrderPurchaseReportOptions
/path/OrderPurchaseReportProduct.pdf

Figure 7: Actual OFBiz Scenario

CBMG for IP: [192.168.1.230] and Session_id: [3]		
From State [State ID - URL] path == ordermgr/control	To State [State ID - URL] path == ordermgr/control	Think Time [sec]
[-1-None]	[26-/path/main]	[0]
[26-/path/main]	[25-/path/login]	[1]
[25-/path/login]	[18-/path/LookupUserLoginAndPartyDetails]	[1]
[18-/path/LookupUserLoginAndPartyDetails]	[27-/path/orderentry]	[0]
[27-/path/orderentry]	[18-/path/LookupUserLoginAndPartyDetails]	[0]
[18-/path/LookupUserLoginAndPartyDetails]	[16-/path/LookupCustomerName]	[1]
[16-/path/LookupCustomerName]	[16-/path/LookupCustomerName]	[0]
[16-/path/LookupCustomerName]	[16-/path/LookupCustomerName]	[0]
[16-/path/LookupCustomerName]	[24-/path/initorderentry]	[1]
[24-/path/initorderentry]	[30-/path/setOrderCurrencyAgreementShipDates]	[0]
[30-/path/setOrderCurrencyAgreementShipDates]	[17-/path/LookupProduct]	[0]
[17-/path/LookupProduct]	[17-/path/LookupProduct]	[0]
[17-/path/LookupProduct]	[21-/path/additem]	[0]
[21-/path/additem]	[29-/path/quickcheckout]	[1]
[29-/path/quickcheckout]	[17-/path/LookupProduct]	[0]
[17-/path/LookupProduct]	[21-/path/additem]	[0]
[21-/path/additem]	[17-/path/LookupProduct]	[0]
[17-/path/LookupProduct]	[28-/path/processorder]	[1]
[28-/path/processorder]	[22-/path/checkout]	[0]
[22-/path/checkout]	[31-/path/updateCheckoutOptions/quickcheckout]	[0]
[31-/path/updateCheckoutOptions/quickcheckout]	[20-/path/OrderPurchaseReportProduct.pdf]	[1]
[20-/path/OrderPurchaseReportProduct.pdf]	[19-/path/OrderPurchaseReportOptions]	[0]
[19-/path/OrderPurchaseReportOptions]	[23-/path/getAssociatedStateList]	[0]

Figure 8: Sample Derived CBMG

Session ID	URL path == ordermgr/control	Number of Visits
3	/path/LookupCustomerName	3
3	/path/LookupProduct	4
3	/path/LookupUserLoginAndPartyDetails	2
3	/path/OrderPurchaseReportOptions	1
3	/path/OrderPurchaseReportProduct.pdf	1
3	/path/additem	2
3	/path/checkout	1
3	/path/getAssociatedStateList	1
3	/path/initorderentry	1
3	/path/login	1
3	/path/main	1
3	/path/orderentry	1
3	/path/processorder	1
3	/path/quickcheckout	1
3	/path/setOrderCurrencyAgreementShipDates	1
3	/path/updateCheckoutOptions/quickcheckout	1

Figure 9: Number of Visits for the URLs in Session 3 CBMG

Session ID	OFBiz Workflow	Total Response Time (in Seconds)
1	/accounting/	2.8
2	/humanres/	1.24
3	/ordermgr/	3.54
4	/accounting/	2.67
5	/humanres/	1.18
6	/ordermgr/	3.82
7	/accounting/	1.69
8	/humanres/	1.09
9	/ordermgr/	2.81
10	/accounting/	2.45
11	/humanres/	1.17
12	/ordermgr/	3.84
13	/accounting/	1.48
14	/humanres/	1.14
15	/ordermgr/	3.11
16	/accounting/	2.49
17	/humanres/	1.09
18	/ordermgr/	3.82
19	/accounting/	1.41
20	/humanres/	1.1
21	/ordermgr/	4.07
22	/accounting/	2.18
23	/humanres/	2.42
24	/ordermgr/	3.73
25	/accounting/	1.45
26	/humanres/	1.22
27	/ordermgr/	3.96
28	/accounting/	1.67
29	/humanres/	1.94
30	/ordermgr/	3.63

Figure 10: Total Response Time Per Workflow Per Session

OFBiz Workflow	Average Response Time (in Seconds)
/accounting/	2.029
/humanres/	1.359
/ordermgr/	3.633

Figure 11: Average Response Time Per Workflow Across Sessions

(or tables) by changing the request parameters. When parsing Apache access logs where the URL has been requested using the POST method, one would find the same URL repeated multiple times even though the URL was used to serve different purposes. The dynamic performance model derivation framework that we are attempting to implement is meant to be non-intrusive. Short of using intrusive methods for capturing request parameters in POST HTTP requests, we find no other way but to group similar URLs based on the module name as opposed to the parameters used in the request.

4. When trying to infer user behavior from system logs, it is important to make some simplifying assumptions since the objective here is not to completely and accurately understand the user's behavior. Instead, the objective is to derive a workload model, which is -by definition- an abstraction of the actual system. In our case, we used JMeter to simulate 10 users accessing three different OFBiz ERP applications. The simplifying assumption we made is that the same user can not work on two tasks simultaneously. Although it might be possible to use multiple browsers or multiple computers at the same time, we assumed that a user can either enter a product order, issue a payment or update an employee's skill set at any given time.

7 Related Work

In [11], the authors implemented the same workload characterization algorithm described in [8, 9] for the purpose of generating similar workloads in stress testing tools (i.e., load testing) to answer what-if questions. The authors developed a graphical user interface to aid in producing the CBMGs and generating stress testing workloads. Although the authors did not highlight the technology stack used to implement their tool, we feel that the use of the ELK stack and Logstash in particular makes our approach more generic and more efficient. For example, in our experiments we implemented automated log parsing of Apache common log format as well as combined log format plus Tomcat access logs with a slight modification of the Grok filter. The use of Elasticsearch for storing log entries allows for fast search and processing, which is essential when parsing and analyzing multiple large log files. Finally, the purpose of our approach for parsing server logs is to reverse engineer the systems into their corresponding QN performance models. This requires that we parse system access logs as well as other system activity logs. The use of the ELK stack allows us to use a consistent tool set to parse and analyze system logs, and derive the corresponding performance model.

In [4], the authors presented Modellus; a system for automated web-based application modeling that uses workload characterization, data mining and machine learning techniques. Their approach is similar to the automated framework for the dynamic derivation of analytical performance models in autonomic systems described in [2], which is the framework we are starting to implement in this paper. The authors use learning techniques from artificial intelligence networks, and use a Directed Acyclic Graph similar to the state diagram used in CBMGs to show how transactions split, join or are linked sequentially. Modellus uses a monitoring agent that is deployed on the production system and used to monitor the host system and report monitoring data. In our approach, we rely on analyzing standard system logs without installing any agents on the host system, which we consider an intrusive method especially in the case of autonomic systems, such as cloud computing providers, where it may not be feasible to coordinate with the systems owners to install such agents. Also, Modellus uses access logs in order to run statistical analysis for the purpose of modeling the linear aspects of computer system and is not effective in modeling non-linear aspects of the host application system, such as response time. Finally, our use of the ELK stack makes our approach more generic and prepares it to be extended to parsing other system logs.

In [12] the authors proposed a method to more accurately match the transaction arrival rates with intervals of steady resource utilizations. By comparing the service demand measurements at different intervals of steady resource utilizations, one can continue to improve the service demand estimates. The Workload Modeler (WLM) tool parses application access logs for the sole purpose of finding arrival rates in order to compare them with server utilization logs. Our approach performs both syntactic analysis and semantic analysis by extracting complete workflows from the access logs. We focus on modeling application systems by analyzing the user interaction with the system and determining workloads accordingly as opposed to analyzing requests individually.

In [7], the authors presented a survey of performance modeling approaches focusing mainly on business information systems. The authors described the general activities

involved in workload characterization, especially estimating service demands, and the various methods and approaches used to estimate it. Some of these methods include general optimization techniques, linear regression, and Kalman filters

In [6], the authors presented an on-line workload classification mechanism for optimized forecasting method selection. Based on the level of overhead (noise level) presented in the various time-series-based workload classification methods, the proposed mechanism dynamically selects the appropriate method using a decision tree. The decision tree takes into account user specified forecasting objectives. The approach presented in this paper can be combined with our proposed framework to forecast future arrival rate for better workload clustering and model derivation.

A similar objective is addressed in [5], where the authors proposed a trace-based approach for capacity management. Workload pattern analysis uses a pattern recognition step and quality and classification step, which measure the difference between the current workload and the pattern, computed as the average absolute error. The authors also presented a process for generating a synthetic trace to represent future workload demands.

In [3], the authors presented a survey of the various workload modeling techniques, such as clustering for interactive systems and the use of graphs (directed graphs, communication graphs and task graphs) for parallel systems. The authors identified the different workload model characteristics (parameters) for each of the representative system in the survey and presented the steps needed to construct a workload model.

8 Conclusions and Future Work

We have presented an implementation approach for automatically deriving CBMGs using system logs. Specifically, we used the ELK stack to parse Apache Tomcat access logs and to infer the repeatable user behavior, which is transformed into CBMGs. We ran a number of experiments on Apache OFBiz ERP system and used Logstash to parse the Tomcat access logs. The results show that our approach for deriving CBMGs is accurate and can be used to estimate the various system workloads.

Similar to the process used in this paper for automatically parsing and analyzing the webserver access logs, we plan to automate the process of parsing and analyzing application server logs and database logs. We are currently developing the templates and configuration files for parsing Oracle WebLogic application server logs and Oracle database redo logs. We are also investigating various ways to derive the system's infrastructure model type (the system's queues and its interconnections.) The outcome of this analysis will be used to implement a framework for automatically deriving system performance models from systems running in production without requiring much prior knowledge of the system's infrastructure or application system behavior.

References

[1] Awad, M. and Menascé, D. A., "On the Predictive Properties of Performance Models Derived Through Input-Output Relationships", Proceedings of the European Performance Engineering Workshop (EPEW 2014), 2014.

[2] Awad, M. and Menascé, D. A., "Dynamic Derivation of Analytical Performance Models in Autonomic Computing Environments", In: Computer Measurement Group Conference on Performance and Capacity, 2014.

[3] Calzarossa, M.; Serazzi, G., "Workload characterization: a survey," Proceedings of the IEEE , vol.81, no.8, pp.1136,1150, 1993.

[4] Desnoyers, P., Wood, T., Shenoy, P., Singh, R., Patil, S., and Vin, H.: Modellus: Automated modeling of complex internet data center applications. In: ACM Tr. on the Web (TWEB) 6, no. 2 (2012).

[5] Gmach, D., Rolia, J., Cherkasova, L. and Kemper, A., Workload Analysis and Demand Prediction of Enterprise Data Center Applications. In: Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization, pp. 171–180, 2007.

[6] Herbst, N.R., Kounev, N.S., and Amrehn, E., Self-Adaptive Workload Classification and Forecasting for Proactive Resource Provisioning. In: Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering, 187–198, 2013.

[7] Kounev, S., Huber, N., Spinner, S., Brosig, S., Model-Based Techniques for Performance Engineering of Business Information Systems. In: Business Modeling and Software Design, Lecture Notes in Business Information Processing (LNBIP), Vol. 0109, pp. 19–37, 2012.

[8] Menascé, D.A., Almeida, V., Fonseca, R. and Mendes, M.A., A Methodology for Workload Characterization of E-commerce Sites, In: ACM Conference on Electronic Commerce, 1999.

[9] Menascé, D., Almeida, V., Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning. Prentice Hall, 2000.

[10] Menascé, D., Almeida, V., and Dowdy, L., Performance by Design: Computer Capacity Planning By Example. Prentice Hall, 2004.

[11] Ruffo, G. , R. Schifanella, M. Sereno, and R. Politi, WALTy: A User Behavior Tailored Tool for Evaluating Web Application Performance. In: Proceedings of the

third IEEE International Symposium on Network Computing and Applications (NCA04), 2004.

[12] Sudheendra, Suhas, Mitesh Patel, and Pratik Kumar. "Approach to build performance model for a web-based system from its application server logs." In: Proceedings of the 32nd International Computer Measurement Group Conference, 2006.

[13] Ghanbari, H., Barna, C., Litoiu, M., Woodside, M., Zheng, T., Wong, J., Iszlai, G., Tracking Adaptive Performance Models Using Dynamic Clustering of User Classes. In: Proceedings of the 2nd ACM/SPEC International Conference on Performance engineering, 2011.