ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 1, Issue 7, September 2012

The Weighted Average Method for Predicting the Performance of Systems

P.Divya Jyothi, S.Vasundra

Abstract— **Predicting performance of a software system with bursts of customer sessions is essential to ensure scalability and availability and manageability. The tools that can predict performance can help in supporting systems in terms of planning, management and system sizing. We propose a new algorithm for predictive performance. That is WAM (Weighted Average Method) which is meant for improving accuracy of predictions. WAM considers bursts of customer requests in order to reflect the influence of bursts in the system. The experimental results revealed that WAM is efficient in predicting performance of selected software system with bursts of customer requests.**

***Index Terms*—-Performance,WAM.**

I. INTRODUCTION

Performance prediction plays an important role in web based applications where number of users uses applications concurrently. For each user a separate session is maintained by web server. However, when number of sessions get increased drastically, performance of applications may slowdown or even the applications may crash. This leads to failure of important applications over web. Though there are many performance prediction applications available, the accuracy of such prediction plays an important role. The aim of this paper is to improve such accuracy in case of applications with bursts of customer sessions. The performance predictions which are accurate can help performance engineers to analyze the performance of applications and take steps pertaining to management, sizing and capacity planning. System responsiveness helps engineers take necessary decisions on the existing applications. Researches made in [1] clarify certain this regarding session length and details pertaining to sessions. According to that the number of requests that come from a user session are considered session length. We believe that just known the number of concurrent sessions is not sufficient to have performance predictions. In addition to this distribution of such concurrent customer sessions is required. This we call as session population distribution. According to studies [2], [3] multitier systems with session based approaches can have bursts of customer sessions that affect the performance of such systems. In order to predict performance of such systems studies have been made in [2] and [4]. However, these studies are not general in nature and not suitable for the problem we considered in this paper. In [5] a hierarchical models which combines the models such as QNM and markov chain birth-death process was proposed.

However, these models are not sufficient in order to find the actual impact of burst of customer sessions with varying parameters. In this paper we propose and evaluate an approach known as WAM (Weighted Average Method) to accurately predict the performance of applications with bursts of customer sessions. This technique applies hybrid modeling with the combination of analytical and simulation models. It has been motivated by the Markov chain model that is best used to estimate population distribution.

II. RELATED WORK

This paper focuses on bursty behavior of concurrent user sessions in case of multi-tier session – based applications such as e-commerce applications. The review of literature in this regard is described here. The workloads with different number of requests can be called as "bursty". Each session is nothing but a series of associated requests from a particular customer or client. It does mean that all requests coming to web server from a particular browser can be called as a session. Each session is a collection of related and associated requests. Session management is the corner stone of any enterprise application where the performance is given more importance. Given this fact, it is essential to have applications to withstand bursts of customer sessions. In [6] Whitt and Bondi presented the impact of high service time difference on the performance of network by using analytical model for performance prediction. In [7] Eager et al. described a technique for solving high service variability problem with a closed network of queues. In [8] Casale et al. recently presented an analytical model to know the performance of applications with bursts of customer sessions. However, addressing service time availability is not the main focus of this work. Compared with this WAM, the proposed algorithm is straight forward and best used to improve accuracy in prediction model that is meant for predicting performance of applications with bursts of customer sessions. According to [2] the session based enterprise applications exhibit problems due to the bursty nature of customer sessions over Internet. This leads to adverse performance of such enterprise applications. An auction system and an e-commerce systems and their workloads are tested in [2] by Mensce et al. The both systems are observed with bursty customer sessions. They use ON-OFF process and its properties to ascertain the fact the bursts are due to heavy-tailed nature of distribution of sessions. In [9] Vallamsetty et al. noticed bursts in the request arrivals from different sessions in case of e-commerce applications. As per their observations, the applications caused highly variable request resource demands, think times and session lengths leading to bursts in customer sessions and the arrival time scales as described in [10] also. This revealed a fact that

modeling customer behavior in terms of bursts of customer sessions can improve the chances of accurate prediction of performance of applications with bursts of customer sessions. The bursty nature of customer sessions can have major impact on applications as they are subjected to unpredictable workloads over a period of time. This may results in those applications to fail in fair processing of requests coming from customers across the globe. In [11] Crovella and Lipsky made experiments on bursty systems and came to know that heavy-tailed distributions of customer requests resulted in different measures for each and every run.

In [10] the bursty nature of multi-tier session based systems has been confirmed. With workloads taken from benchmark TPC-W, they found that many similar runs of measurements with similar resource demands and expected throughput resulted in different mean response time measurements. This is evident that raises a fact indicating the modeling approaches are to be changed to meet the actual requirements. Many predictive performance models have come into existence. However better modeling approaches came into existence known as QNM with Mean Value Analysis and Markove Chains. Some approximation techniques [8] and [12] also came into existence.

Due to the problems in the existing predictive performance models, we propose a model known as WAM (Weighted Average Method) which considers the not only number of concurrent sessions but distributions of session population in order to accurately measure the performance of multi-timer, session – based applications with bursts of customer sessions. The following section describes WAM algorithm.

III. WAM

The Weighted Average Method (WAM) algorithm helps in improving accuracy of predicting performance of applications with bursts of customer sessions. Especially this kind algorithm is useful to web based applications such as e-Commerce, enterprises. This is because the web applications can be used by multiple users concurrently. Thus concurrent user sessions are possible. This is the suitable environment in which WAM can be applied. The WAM parameters and the procedure are given in fig. 1 in the form an algorithm.

As provided in fig. 1, the WAM algorithm starts with some event list known as FEL (Future Event List). This vector contains a set of request objects. Each request object contains request start time, end time and a Boolean flag indicating whether the request is the last request in a session. These request objects are loaded from historical trace table. Then it initializes all parameters required including aggregate state time array, aggregate state completions array and other variables like state start time, state end time etc. Statement 9 is the while loop that is responsible to iterate through all request of FEL. Finally it computes think time and creates request submission event besides updating FEL. Finally line 13 computes R_{mean} values and they are sent to table. From table they are picked and presented in the form of a graph as shown in fig. 3.

1. Create a Future Event List (*FEL*). *FEL* stores events in chronological order.
2. *Current_Population*=0
3. *State_Start_Time*=0
4. *State_End_Time*=0
5. Initialize elements of *Aggregate_State_Time* array to 0. This array has N_{max} elements where N_{max} is the maximum population.
6. Initialize elements of *Aggregate_State_Completions* array to 0. This array has N_{max} elements.
7. Obtain *Predictive_Model_Response_Time* array by solving a predictive model. This array has N_{max} elements.
8. Create request submission events corresponding to first requests of all sessions in trace *S*. Store the events in the *FEL*
9. **While** *FEL* is non-empty
   Select earliest event in *FEL*
   If event is submission of a request
      If request is first request in a session
         State_End_Time = *start time* of request
         Aggregate_State_Time[*Current_Population*]+=(*State_End_Time*-*State_Start_Time*)
         Completions=Request completions in the period (*State_Start_Time*, *State_End_Time*)
         Aggregate_State_Completions[*Current_Population*]+=*Completions*
         State_Start_Time=*State_End_Time*
         Current_Population+=1
      End If
      If *S* is a historical trace
         Response_Time = Get actual response time of request
      End If
      If *S* is a synthetic trace
         Response_Time = *Predictive_Model_Response_Time*[*Current_Population*]
      End If
      Create a request completion event at (*State_Start_Time*+*Response_Time*)
      Update *FEL* with the event
      Continue
   If event is completion of a request
      If request is last request in a session
         State_End_Time = *end time* of request
         Aggregate_State_Time[*Current_Population*]+=(*State_End_Time*-*State_Start_Time*)
         Completions=Request completions in the period (*State_Start_Time*, *State_End_Time*)
         Aggregate_State_Completions[*Current_Population*]+=*Completions*
         State_Start_Time=*State_End_Time*
         Current_Population-=1
         Continue
      End If
      Think_Time = Get think time of request
      Create a request submission event at (*State_Start_Time*+*Think_Time*)
      Update *FEL* with the event
      Continue
   End While
10. Compute *Total_Time* as sum of elements of *Aggregate_State_Time*
11. Compute P_i values by dividing each element of *Aggregate_State_Time* by *Total_Time*
12. Compute X_i by dividing each element of *Aggregate_State_Completions* with the corresponding element of *Aggregate_State_Time*
13. Use (6) to compute mean response time

Fig. 1 – WAM algorithm

IV.EXPERIMENTS

This section describes the experiments and the results including the environment used for the sake of experiments.

Environment

The environment used for experiments include Java 6.0, NetBeans IDE, Oracle 10G Express Edition running in a PC with Windows 7 Operating System. NetBeans IDE is used for rapid application development while Oracle database is used to store historical trace and also results of experiments.

Experiment Design

Experiment design includes a GUI application that facilitates the demonstration of WAM algorithm that is meant for predicting the performance of an application with bursts of client requests. Java's Swing API is used to build graphical user interface while JDBC API is used to interact with backend. WAM algorithm is implemented using core Java logic based on [22].

Methodology

The methodology used to implement WAM and perform experiments with bursts of customer sessions is

described here. First of all a historical trace is obtained and stored in Oracle database. Therefore the historical trace is considered as dataset for the experiments. The actual experiments are performed using a prototype application with GUI. The operations performed through GUI are generating historical trace, deleting historical trace, viewing historical trace, initializing FEL, building FEL from historical trace, predicting performance and generating graph automatically. The results are saved in database and the graph picks values from table to show the results of WAM algorithm.

Prototype Application

The frontend application built to demonstrate the WAM algorithm for predicting performance of an application with bursts of customer sessions provides user-friendly interface that facilitates all operations. Fig. 1 shows the main UI screen.

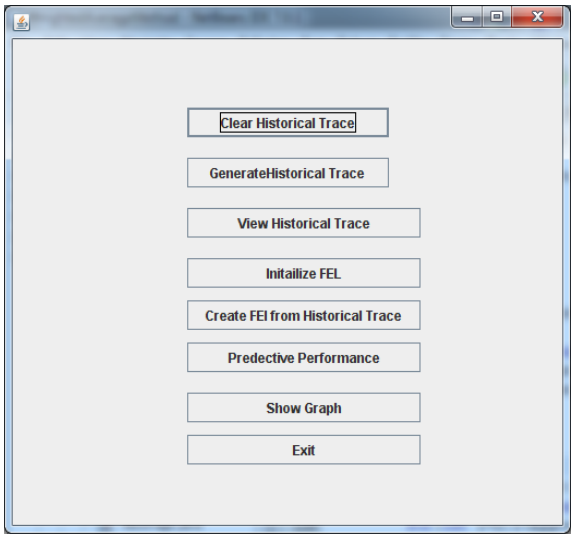

Fig. 2 – Main Screen with WAM related operations

As can be seen in fig. 2, the operations supported by the frontend application include clearing historical trace, generating historical trace, viewing historical trace, initializing FEL, creating FEL from historical trace, predicting performance of given application based on the dataset (historical trace). Then options are given for showing graph and terminating the frontend application. On choosing "Clear Historical Trace" button, it automatically deletes all records in the historical trace table. This is required to experiment the application with different number of bursts of customer sessions. The "Generate Historical Trace" button is meant for taking number of sessions and number of requests per session as input from user and generates records into historical trace table. These records in the table are considered to be dataset for the experiments.

This operation facilitates to have multiple trace related to various applications and apply WAM algorithm to predict performance of the application with bursts of customer sessions. These records are later loaded into RAM for further processing.

The "View Historical Trace" button is meant for viewing historical trace that is automatically generated based on the number of sessions and requests of each session as input given by the user. This option facilitates end users to have an idea of the trace before conducting actual experiments with WAM algorithm. The "Initialize FEL" button is meant for initializing Future Event List. Once FEL is initialized it is possible to populate FEL with actual events provided in the form of historical trace. The "Create FEL from Historical Trace" button is responsible to take records from historical trace file and load each record into a POJO (Plain Old Java Object) object and then has a java.util.List object containing a collection records that constitute FEL. By now the historical trace with all events are readily available in FEL. This is the time to apply WAM algorithm shown in Fig. 1 on FEL. The "Predict Performance" button is meant for actually performing the proposed algorithm on FEL and provides results. This kind of experiments is made number of times and the results are recorded.

V. RESULTS

As can be seen in fig. 3, the CDF (Cumulative Distribution Function) varies for different replications. The summary of all experiments are presented in the graph. In X axis replications are presented while the Y axis shows response time.

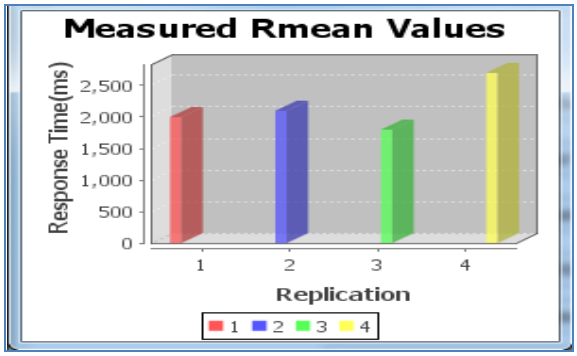

Fig. 3 – Measured R_{mean} values for given dataset

The replication 4 shows longest response time while the replication 3 shows shortest response time. The highest R_{mean} value and lowest R_{mean} value are exhibited by CDFs of replication four and three respectively. These results from WAM algorithm can help application performance analysts to estimate the application performance with bursts of customer sessions. They can easily predict the problems and provide solutions with ease by repeatedly running various applications with different session traces. This is the significance of WAM algorithm explored with a prototype application in this paper.

VI. CONCLUSION

In this paper, we present and explore an algorithm by name Weighted Average Method (WAM) for enhancing the accuracy in predicting the performance of applications with bursts of customer sessions. For all session based systems especially web applications including e-commerce

and distributed enterprise applications the proposed technique can be applied and the performance analysts can make decisions based on the predictive performance of application done using WAM. WAM allows experiments to be made with real world traces of customer sessions and also the traces synthesized. We developed a prototype application with GUI to demonstrate the effectiveness of WAM. We generated session traces for the sake of experiments. We also tested the application with various traces and the summary of values is presented. The experimental results revealed that the WAM algorithm is able to give accurate results and this is useful in predicting performance of real world applications with bursts of customer sessions.

REFERENCES

[1] T. Bonald and J.W. Roberts, "Congestion at Flow Level and the Impact of User Behaviour," Computer Networks, vol. 42, pp. 521- 536, 2003.

[2] D. Menasce, V. Almeida, R. Reidi, F. Pelegrinelli, R. Fonesca, and W. Meira JR, "In Search of Invariants in e-Business Workloads," Proc. ACM Conf. Electronic Commerce, pp. 56-65, Oct. 2000.

[3] U. Vallamsetty, K. Kant, and P. Mohapatra, "Characterization of e-Commerce Traffic," Electronic Commerce Research, vol. 3, nos. 1/2, pp. 167-192, 2003.

[4] K. Psounis, P. Molinero-Ferna´ndez, B. Prabhakar, and F. Papadopoulos, "Systems with Multiple Servers under Heavy-Tailed Workloads," Performance Evaluation, vol. 62, nos. 1-4, pp. 456- 474, Oct. 2005.

[5] D. Menasce and V. Almeida, Capacity Planning and Performance Modeling: From Mainframes to Client-Server Systems. Prentice Hall, 1994.

[6] A.B. Bondi and W. Whitt, "The Influence of Service-Time Variability in a Closed Network of Queues," Performance Evaluation, vol. 6, no. 3, pp. 219-234, Sept. 1986.

[7] D.L. Eager, D.J. Sorin, and M.K. Vernon, "AMVA Techniques for High Service Time Variability," Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems, pp. 217-228, 2000.

[8] G. Casale, N. Mi, and E. Smirni, "Bound Analysis of Closed Queuing Networks with Workload Burstiness," Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems, pp. 13-24, 2008.

[9] U. Vallamsetty, K. Kant, and P. Mohapatra, "Characterization of e-Commerce Traffic," Electronic Commerce Research, vol. 3, nos. 1/2, pp. 167-192, 2003

[10] D. Krishnamurthy, J. Rolia, and S. Majumdar, "A Synthetic Workload Generation Technique for Stress Testing Session-Based Systems," IEEE Trans. Software Eng., vol. 32, no. 11, pp. 868-882, Nov. 2006.

[11] M.E. Crovella and L. Lipsky, "Long-Lasting Transient Conditions in Simulations with Heavy-Tailed Workloads," Proc. 29th Conf. Winter Simulation, pp. 1005-1012, 1997.

[12] Y. Chen, S. Iyer, X. Liu, D. Milojicic, and A. Sahai, "SLA Decomposition: Translating Service Level Objectives to System Level Thresholds," Proc. Fourth Int'l Conf. Autonomic Computing, 2007.

P.Divyajyothi M.Tech Student, CSE Department, JNTUACEA, Anantapur, Andhra Pradesh, India.

S.Vasundara$_{M.Tech,(Ph.D.)}$,Associate Professor, CSE Department, JNTUACEA, Anantapur, Andhra Pradesh, India.