Audit Trail Analysis for Traffic Intensive Web Application

Ka-I Pun, Yain-Whar Si
Faculty of Science and Technology, University of Macau
{ma56543, fstasp}@umac.mo

Abstract – **Web-enabled business processes designed for travel industry and government agencies are considered as traffic intensive applications since number of users can fluctuate dramatically within a short time. Applications under this category are likely to encounter flash crowd situation in which the server can no longer handle overwhelming service requests. To alleviate this problem, business process engineers usually analyze audit trail data recorded from the application server and reengineer their processes to withstand such situations. However, audit trail analysis can only reveal some of the performance indicators which can only be observed from the internal perspective of the application server. In this research, we propose a novel approach for identifying key performance indicators of traffic intensive web applications by integrating analysis results from audit trail data of an application server with analysis results from a web server log analyzer and stress testing tool. Web server log analyzers are mainly used to analyze the transactions between client computers and the web server whereas stress testing tools are usually used to estimate the workload limit of a web server. The analysis result from these programs provides an internal as well as an external view of the application performance. Such integration allows process engineers to exactly pinpoint the potential bottlenecks in the application.**

I. INTRODUCTION

Service industries, government portals, and e-commerce companies deploy various kinds of web applications for delivering services to their customers. These applications are often modeled as business processes and powered by Workflow Management Systems (WfMS). A workflow management system is "*a system that completely defines manages and executes workflow through the execution of software whose order of execution is driven by a computer representation of workflow logic*" [3]. It is often considered as a procedural automation of a business process based on process-oriental model. However, the service demand on a web application can spike rapidly when unanticipated events occur. For instance, situations such as extreme weather, political crisis, and sudden surge in the popularity of the web site can lead to unexpected increase in the number of service requests. We label these applications as *traffic intensive* since they are vulnerable to fluctuation in the number of service requests within a short time. Traffic intensive web applications which are commonly used in travel industry and government agencies for providing services are potentially susceptible to *flash crowd* [1] situation. A flash crowd is a surge of unanticipated, rapid, huge number of simultaneous requests, and accesses to a particular website.

During flash crowd situation, servers which are used to host traffic intensive web applications might be unreachable since they can no longer handle overwhelming service requests. To alleviate this problem, business process engineers usually analyze the audit trail data to pinpoint the bottleneck locations in these workflows. An audit trail is "*an electronic archive in which the history of a workflow is recorded*" [4]. It records the changes of state of each process in the workflow as well as the interactions between application server and client computers at operational level chronologically. However, analysis performed on audit trail data of an application server can only provide an internal view of the situation. Specifically, analysis of audit trail can only reveal resource allocation as well as the history of execution of a business process at an operational level. Information such as time required for the web server to respond to each hit requested from a thread and results of the requests (success or fail) made by the threads are not available in audit trail data. Such information is considered as external to the application under evaluation.

In this research, we propose a novel approach for identifying key performance indicators of traffic intensive web application by integrating analysis results from audit trail data of application server with analysis results from web server log analyzer and stress testing tool. Web server log are usually used to record the history of page requests of a web server within a specified duration. Information contained in a web server log typically includes the access records from client computers as well as the error records of each page requests. Web server log analyzers are mainly used to analyze the transactions between client computers and the web server. In addition to web server analysis, web stress testing tools are often used to evaluate the workload limit of a web server. They are also used to analyze the performance of the web application in handling massive concurrent page requests. In contrast to web server log, results from stress testing tool provide a form of evaluation from client perspective (external view). Therefore, integrating analysis results from audit trail data of application server with analysis results from web server log analyzer and stress testing tool provides internal as well as external view of the application performance. Such integration allows process engineers to exactly pinpoint the potential bottlenecks in the application.

This paper is organized as follows. Section II presents an overview of the approach. A formal analysis of the requirements of an audit trail analyzer for extracting the statistics is given in section III. We briefly describe the analysis on the statistics from web server log analyzer and stress testing tools in section IV and V. Section VI integrates the extracted analysis results and distinguishes the critical performance indicators. We discuss the relevant research work in section VII and conclude our ideas in section VIII.

II. OVERVIEW

Fig. 1 conceptualizes the integration of the analysis tools which are used to perform a comprehensive evaluation of an application server from different perspectives. Both audit trail and web server log analyzer provide an internal/server view of an application server. Audit trail analyzer can be designed to utilize the business logic of the workflow whereas web server log analyzer is capable of depicting transaction details between the application server and client computers. On the other hand, stress testing tool provides an external/client view of an application server. Stress testing tool assesses the capability of the application in handling unexpected massive page requests as well as estimates its workload limit.

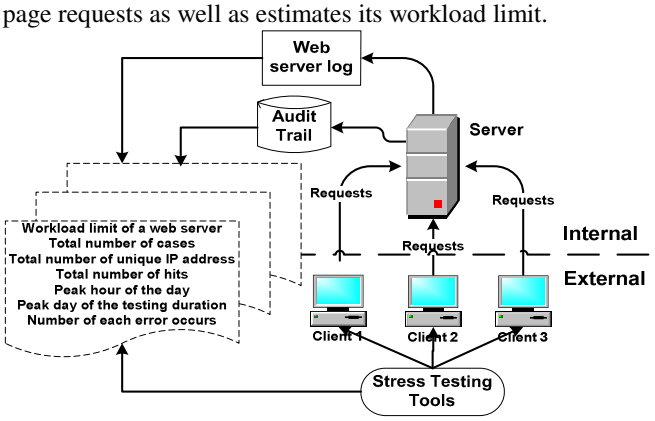

Fig. 1. Conceptual model of integration of analysis result from audit trail analyzer, web server log analyzer and stress testing tool

As depicted in Fig. 1, a stress testing tool can imitate client computers in generating huge number of simultaneous requests to the server (of a web application). Stress testing tools can also capture the statistics regarding the responses of a web server. In addition, web server itself and the workflow management system are capable of logging the access records of clients and the activities of the application workflow. In this research, we propose an approach on integration and analysis of these data for identifying and locating key performance indicators. These indicators are crucial in understanding of the behavior and stability of the traffic intensive web application.

III. AUDIT TRAIL ANALYZER

In general, audit trail data captured in WfMS is in unprocessed state. The Workflow Management Coalition (WfMC) proposed the interface 5 "Audit Data Specification" [2] which states the audit trail information for a workflow management system. In this section, we briefly describe the design of the audit trail data analysis module. Audit trail chronologically records the resource allocation, as well as the history of execution and result of a business process at operation level. A WfMS captures the workflow model of a business process by defining a *process definition* which consists of a set of *activities*.

During the execution of the WfMS, an initiation of a process definition by the workflow engine will create a *process instance*. Each process instance contains a set of *activity instances*. These activity instances are executed one by one, or sometimes in parallel to accomplish the process instance. Whenever an activity instance is started, one or more related *workitems* will be generated. After a workitem is created, it will be allocated to a resource in the WfMS for execution. Once the workitem is allocated, it is appended at the end of the queue of the assigned resource. These transition states for process instance, activity instance and workitem are recorded in audit trail. We conceptualize the above relation in Fig. 2.

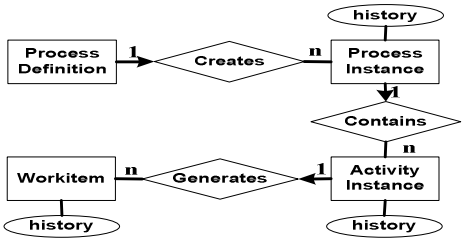

Fig. 2. Relation between process definition, process instance, activity instance, and workitem

During the execution of a workflow, a workitem can be in one of the eight possible states (*enabled, executing, complete, forceComplete, cancelled, fired, deadlocked, isParent*). The lifetime of a workitem can be defined as a set of tuples consisting of states and time points:

$$\{(s_1, start_1, end_1), \ldots, (s_n, start_n, end_n)\},$$

where s_i, $start_i$ and end_i are the state, start time, and end time of the *ith* state in the life time of a workitem.. We can define the lifetime of both activity instance and process instance in a similar way. We define the following notations:

W = the history of the states in the lifetime of a workitem
w = a tuple in W
A = the history of the states in the lifetime of an activity instance
a = a tuple in A
P = the history of the states in the lifetime of a process instance
p = a tuple in P

Functions for accessing the elements of the tuples are defined in Table 1.

	Workitem	Activity Instance	Process Instance
s	wstate((s,start,end))	astate((s,start,end))	pstate((s,start,end))
start	wstart((s,start,end))	astart((s,start,end))	pstart((s,start,end))
end	wend((s,start,end))	aend((s,start,end))	pend((s,start,end))

Table 1. Functions for accessing the element from the tuples of process instance, activity instance and workitem.

Workitem

Based on these functions we extract life duration, allocation time, and processing time of a workitem as follows:

Life duration: The duration of any state in the lifetime of a workitem can be defined as follows:

$$wduration(w) = wend(w) - wstart(w) \quad \textbf{(1)}$$

The life duration of a workitem is the sum of the duration of each state in the history of lifetime.

$$wlife(W) = \sum_{w \in W} wduration(w) \quad \textbf{(2)}$$

Allocation time: A workitem is in enabled state when it is created but not yet allocated to any resources. Allocation time of a workitem is the total duration of the workitem in enabled state.

$$wallocate(W) = \sum_{w \in W \wedge wstate(w) = enabled} wduration(w) \quad \textbf{(3)}$$

Total queuing time: After a workitem is allocated to a resource, the state of the workitem is changed into *fired*. It may need to queue until the resource is available. During the lifetime, the workitem may have more than one fired state since it can be reallocated to other resources. The total queuing time of a workitem is the sum of the duration of the workitem in the fired state.

$$wqueue(W) = \sum_{w \in W \wedge wstate(w) = fired} wduration(w) \quad \textbf{(4)}$$

Processing time: The processing time of a workitem is the total amount of time when it is in an executing state.

$$wprocess(W) = \sum_{w \in W \wedge wstate(w) = executing} wduration(w) \quad \textbf{(5)}$$

Idle time: The idle time of a workitem is the total amount of time when it is not in an executing state.

$$widle(W) = \sum_{w \in W \wedge wstate(w) \neq executing} wduration(w) \quad \textbf{(6)}$$

Activity Instance

Life duration: The duration of any state in the lifetime of an activity instance can be defined as follows:

$$aduration\ (a) = aend\ (a) - astart\ (a) \quad \textbf{(7)}$$

The life duration of an activity instance is the sum of the duration of each state in the history of lifetime.

$$alife(A) = \sum_{a \in A} aduration\ (a) \quad \textbf{(8)}$$

Processing time: The processing time of an activity instance is the total amount of time when it is in an executing state.

$$aprocess(A) = \sum_{a \in A \wedge astate(a) = executing} aduration(a) \quad \textbf{(9)}$$

Idle time: The idle time of an activity instance is the total amount of time when it is not in an executing state.

$$aidle\ (A) = \sum_{a \in A \wedge astate\ (a) \neq executing} aduration\ (a) \quad \textbf{(10)}$$

Process Instance

Life duration: The duration of any state in the lifetime of a process instance can be defined as follows:

$$pduration\ (p) = aend\ (p) - astart\ (p) \quad \textbf{(11)}$$

The life duration of a process instance is the sum of the duration of each state in the history of lifetime.

$$plife(P) = \sum_{p \in P} aduration\ (p) \quad \textbf{(12)}$$

Processing time: The processing time of a process instance is the total amount of time when it is in an executing state.

$$pprocess(P) = \sum_{p \in P \wedge pstate(p) = executing} pduration(p) \quad \textbf{(13)}$$

Idle time: The idle time of a process instance is the total amount of time when it is not in an executing state.

$$pidle(P) = \sum_{p \in P \wedge pstate(p) \neq executing} pduration(p) \quad \textbf{(14)}$$

Based on these formulas, we are able to derive life duration, allocation time, total queuing time, processing time and idle time of workitems, activity instances, and process instances. In addition, audit trail analyzer also retrieves the statistics about the states (successful or cancelled) and distribution of the number of process instances, activity instances and workitems for the recorded period. Statistics retrieved from audit trail analyzer is summarized in Table 2.

Statistics from audit trail analyzer	
A1	Number of successful process instances during the log period
A2	Number of cancelled process instances during the log period
A3	Number of process instances of a process definition created during the log period
A4	Number of activity instances of a process definition created during the log period
A5	Number of workitems of a process definition generated during the log period
A6	Number of resources assigned to handle the process instances created from a process definition during the record period
A7	Life duration of a process instance
A8	Idle time of a process instance
A9	Processing time of a process instance
A10	Life duration of an activity instance
A11	Idle time of an activity instance
A12	Processing time of an activity instance
A13	Life duration of a workitem
A14	Queuing time of a workitem
A15	Allocation time of a workitem
A16	Idle time of a workitem

A17	Processing time of a workitem
A18	Number of process instances, activity instances & workitems in each hour during the log period, this entry identifies the peak hour over the log period
A19	Number of process instances, activity instances & workitems in each day during the log period, this entry presents the peak day over the log period
A20	Time duration of each state of a workitem

Table 2: Statistics from audit trail analyzer

IV. WEB SEVER LOG ANALYZER

Services provided by a web server are generally delivered through *pages* (e.g. .html, .jsp). A *hit* (or a *request*) from a client computer can be an attempt to access any object (e.g. .jpg) from the web page. A *click* to a web page may create an event which starts the relevant activities in the process definition of the workflow of a web application. These activities may lead to multiple hits to the web server. Many accesses to a page can be made from one unique IP address. Some accesses are grouped into a *session* which lasts for a specific duration of time. In a web log analyzer, a session is referred by a *visit*, and the unique IP address associated with a visit is called a *visitor*. A visitor may initiate multiple visits and a visit may include accessing of multiple pages.

A web server log can save the above information as the *history of page requests* recorded during the *web server log period*. The history contains the general information about the transaction between a web server and client computers, including the IP address of the client computers, timestamp of a page request, objects accessed, and so on. W3C [5] standardizes the general format of a web server log. An analyzer (e.g. AWStats Web log analyzer [6]) can be used to parse the log file so as to derive the statistics of the activities between the server and client computers. Table 3 summarizes the performance indicators reported by a web server log analyzer.

Web server log analyzer not only reports the general statistics about page requests, but also provides an internal view of the server. Web log analyzer identifies the rush hours of a day, as well as the peak day within the web server log period in which flash crowd may occur. In addition, web log analyzer also indicates if a page request is successful or not by labeling a HTTP status code (e.g. 404) [16]. By combining the indication of peak hours with the status of each page request, we can observe the performance of the web server during flash crowd situation. This information is crucial in analyzing the performance of a traffic intensive web application in which flash crowds occur frequently. Based on the analysis result, webmasters may consider allocating more resources (e.g. bandwidth, shutting down non-critical functions) to the application server.

Statistics from web server log analyzer	
L1	Web server log period
L2	Number of unique visitors recorded in the log period
L3	Number of visits recorded in the log period
L4	Number of pages recorded in the log period
L5	Number of hits/requests recorded in the log period
L6	Average number of hits for each page recorded in the log period
L7	Number of unique visitors in each day of the log period
L8	Number of visits in each day of the log period
L9	Number of pages accessed in each day of the log period
L10	Number of hits made in each day of the log period
L11	Number of unique visitors accessed in each hour of the log period
L12	Number of pages accessed in each hour of the log period
L13	Number of hits made in each hour of the log period
L14	Number of visits for various session range (e.g. 5 min – 15 min)
L15	Number of times of a page is viewed in the log period
L16	Number of hits with various HTTP status code recorded in the log period
L17	Number of hits with 404 error HTTP code recorded for all pages in the log period
L18	Number of pages a unique visitor accessed in the log period

Table 3: Statistics from web server log

V. STRESS TESTING TOOL

While web server logs provide an internal view of the performance of a web server, statistics from stress testing tools present an external view. *Load testing tools* [7] are widely used to assess the performance of a web server in handling expected workload by running a specified set of scripts emulating the behavior of customers. *Stress testing* is a load testing in which the load placed on the server is raised beyond the usual level. Moreover, stress testing tools can be used to estimate the workload limit of a web server. It is also used to evaluate the capability of a web server in handling massive simultaneous requests.

Stress testing tools can simulate flash crowd situation by generating a large number of services requests (*concurrent connections*) simultaneously. The connections are created from one or more unique IP addresses (*test clients*). Each connection is controlled by a *thread*. A thread can mimic to be a real user in requesting services from a web application. Similar to a web server log, a *hit* (or *request*) in a stress testing tool can be an attempt to access any object (e.g. .jpg) from the web page. Some of the statistics reported from stress testing tools are summarized in Table 4. For instance, stress testing tool analyzes the performance of a web server by recording the response time of the web server for each hit. It also summarizes the results of the requests (success or fail) made by the threads during the testing period.

Statistics from stress testing tool	
S1	Run length
S2	Number of unique test clients

S3	Number of hits with various HTTP status codes (e.g. 404) during the stress testing
S4	Number of concurrent connections
S5	Number of hits/requests
S6	Number of requests per second
S7	Average time taken for the server to respond

Table 4: Statistics from stress testing tool

VI. INTEGRATION

By integrating the analysis results, we are now able to pinpoint some of the crucial performance indicators for traffic intensive web applications. These performance indicators are outlined in following paragraphs.

Workload limit with stable performance (A1, A2, L2, L3, L4, L5, S2, S4, S5, S7): Statistics from web server log (L2, L3 L4 and L5) provide an internal view of the performance of the web server whereas statistics from a stress testing tool (S2, S4 and S5) provides an external view. In general, changes in number of pages visited and hits made will proportionally increase or decrease the number of process instances and activity instances (A1 and A2). However, during flash crowd situation, massive number of concurrent requests may reverse the proportion. In this case, time taken to respond the incoming requests (S7) may increase. In the worst case, the web server will be unreachable because of system failure. This will lead to decrease in the number of successful process instance (A1) and increase in the number of cancelled process instance (A2). By analyzing these entries, we can estimate the load limit of a web server as well as the acceptable level of concurrent connections. Fig. 3 shows an example screenshot of this indicator.

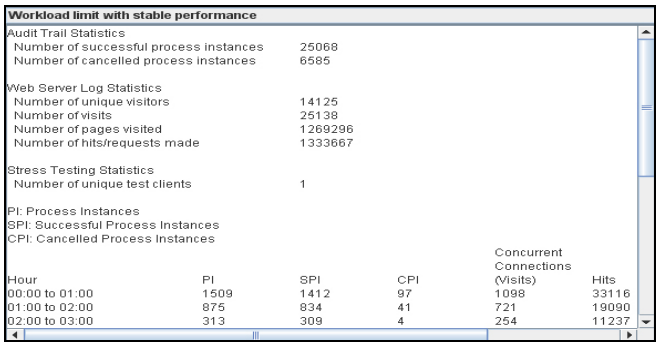

Fig. 3. Workload limit with stable performance

Hourly performance of a system (A18, L11, L12): A18, L11 and L12 can be used to analyze performance of the web server in more detail. Basically, the increase in the number of pages accessed should result similar magnitude of increase in number of process instances and activity instances. However, once the number of pages accessed reaches certain threshold, any decline in the number of process instances and activity instance may indicate that the performance of the web server begins to degrade due to overwhelming number of concurrent accesses. In addition, from L11 and L12, process engineer can determine peak and off-peak hours of the application for rescheduling available resources. Fig. 4 shows an example screenshot of the indicator for hourly performance of a system.

Index	Hour	Process Instances	Activity Instances	Workitems	Pages	Hits/Requests
0	00: 00 to 01:00	1509	16487	38273	30851	33116
1	01:00 to 02:00	875	8273	25183	17500	19090
2	02:00 to 03:00	313	3815	10746	10077	11237
3	03:00 to 04:00	251	2761	8159	4822	5519
4	04:00 to 05:00	207	2087	6245	2599	3058
5	05:00 to 06:00	89	943	2781	1167	1560
6	06:00 to 07:00	73	806	2403	1158	1595
7	07:00 to 08:00	146	1554	6397	2649	3160
8	08:00 to 09:00	327	3452	11911	9558	10898
9	09:00 to 10:00	1656	17397	55453	49211	52176
10	10:00 to 11:00	1935	20652	68165	83258	86580
11	11:00 to 12:00	1981	20173	66736	93090	96796
12	12:00 to 13:00	1697	17874	56912	59459	62104
13	13:00 to 14:00	923	11318	35929	21973	24202
14	14:00 to 15:00	1811	17625	56031	75319	78753
15	15:00 to 16:00	2155	20375	64448	113508	117728
16	16:00 to 17:00	2151	20276	63075	106374	110127
17	17:00 to 18:00	2269	21545	68496	121295	125458
18	18:00 to 19:00	2123	19873	63764	110363	115517
19	19:00 to 20:00	2087	18276	58352	100802	106235
20	20:00 to 21:00	1941	18153	58243	84457	88491
21	21:00 to 22:00	1774	17926	56845	60711	64255
22	22:00 to 23:00	1693	17028	55934	55319	58756
23	23:00 to 00:00	1667	16812	52433	53776	57256

Fig. 4. Hourly Performance of a system

Daily performance of a system (A19, L7, L8, L9, L10): From A20, L7, L8, L9 and L10, we can identify which day of the week or month has the highest recorded traffic. These entries can also reveal any slack or breakdown in the application server operation. Fig. 5 shows the screenshot of this indicator.

Day	Process Instances	Activity Instances	Workitems	Unique Visitors	Visits	Pages	Hits/Requests
2008-09-01	993	10072	30312	403	827	12421	13939
2008-09-02	6120	62405	215732	2265	5006	204686	216456
2008-09-03	6135	68192	220853	2539	5108	257949	271138
2008-09-04	7881	130924	260651	3323	6297	399479	418243
2008-09-05	7721	119576	252438	4124	6256	375433	391177
2008-09-06	2803	5028	12928	1471	1644	19328	22714

Fig. 5. Daily Performance of a system

Problems in process definition (A2, L15, S3): A2 shows the number of total cancelled process instance, while L15 and S3 show various HTTP codes occurred during the performance evaluation. Some of them cause client or server error in accessing the page which may lead to failure in the workflow of a web application. From the client and server error HTTP status codes (e.g. 404), we can determine what kinds of error the server is experiencing and we can apply necessary modifications to the process definition. Furthermore, these records may also indicate frequent occurrence of certain errors.

Identification of cancelled activity instances (A2, L16): A process instance will be cancelled by the web application if the related activity instance fails. When a web server is overloaded, timeout may occur and the client may receive "page not found" response. This failure is usually represented by an HTTP status code (error 404). It also indicates that requested resource could not be found at the moment but may be available again in the future, and therefore subsequent requests to the same page by the client are allowed. By linking A2 and L17, an activity instance can be associated with a corresponding URL of a page in which it is embedded. From such association, we can determine whether a process instance is cancelled due to a certain page can not be found, or due to lack of response from the server. This indicator identifies the potential process instance cancellation in the workflow due to error 404.

VII. Related Work

Analysis on traffic intensive web application with the integration of audit trail with statistics from web server log and stress testing tools has received relatively little coverage in the related literature. To the best of authors' knowledge, recent work in performance evaluation research investigates these statistics in isolation.

Muehlen et al. [8] develop a tool which can be employed to analyze audit trail data of different workflow management systems. Audit trail data is evaluated from three different perspectives (process, resource and object) to monitor and control the workflow of a business process. Compared to their approach, our system not only focuses on the workflow of the web application, but also evaluates the performance of the web application from the internal view and external view of the web server.

Chen et al. [1] introduce an adaptive admission mechanism to maintain the availability of servers. In their approach, analyses on web server logs are performed to detect performance degradation which is considered as a sign for flash crowd situation. Flash crowds are then mitigated by admitting incoming requests adaptively. In contrast to their approach, our framework is only used to analyze the capability of a web application server in handling flash crowd situation.

In [15], Arlitt et al. describe a workload characterization study of the 1998 World Cup Web site. In their work, a cache consistency mechanism is used to reduce the burden in handling the incoming requests. In their approach, frequently accessed objects are placed in cache for faster retrieval by the web clients. However, as web applications are evolving from static to dynamic, and therefore the cache mechanism may not significantly reduce the incoming requests during flash crowd situation. In contrast to their approach, our proposed framework aims to enhance the workload handling capability of a web server by uncovering the weakness of the application workflow.

For load testing, Draheim et al. [9] propose an approach in which users' behavior is mimicked with stochastic form-oriented analysis models. In their approach, web site navigation and time delay are modeled stochastically. Krishnamurthy et al. [10] also propose a technique to generate a synthetic workload for performing stress testing for session-based systems. However, these approaches focus on the techniques used in generating workload for stress loading test, whereas our approach focuses on integration and analysis of the test results with web server log statistics and audit trail data.

VIII. Conclusions

The main contributions of our research work are two-fold; from the theoretical standpoint, we contribute to the analysis, design, and development of an integrated framework including audit trail analyzer, web log analyzer, and stress testing tool. Such integration provides both external and internal view of the evaluation performed on traffic intensive web applications. From the practical standpoint, our research opens the door to the further development of mechanisms for business process reengineering, improved load balancing, and implementation of contingency measures for traffic intensive web applications.

Based on the performance indicators extracted, process engineer may revise the workflow schema of the web application. Such revision could be carried out iteratively until acceptable performance level is achieved. The integrated environment for audit trail analysis was developed based on JAVA [12] programming language with Eclipse [11], YAWL editor [13], open source web analyzer AWStats [6], and Microsoft Web Application Stress Tool [14].

IX. Acknowledgement

This research is funded by the University of Macau under grant RG056/06-07S/SYW/FST.

References

[1] X. Chen and J. Heidemann, "Flash crowd mitigation via adaptive admission control based on application-level observations" 5(3), pp. 532 – 569, in *ACM Transactions on Internet Technology*, 2005.

[2] Workflow Management Coalition, Audit Data Specification, Document Number WFMC-TC-1015, version 1.1, September 1998.

[3] Workflow Management Coalition, Workflow Reference Model, Document Number TC00-1003, Version 1.1, January 1995, http://www.wfmc.org/standards/docs/tc003v11.pdf.

[4] W. M. P. van der Aalst and K. M. van Hee, "Workflow Management. Models, Methods, and Systems", MIT Press, Cambridge, 2002.

[5] W3C, Extended Log File. http://www.w3.org/TR/WD-logfile.

[6] AWStats. http://awstats.sourceforge.net/.

[7] D.A. Menascé, "Load Testing of Web Sites", pp. 70 – 74, in *IEEE Internet Computing*, vol. 6, no. 4, July/August 2002.

[8] M. zur Muehlen and M. Rosemann, "Workflow-based process monitoring and controlling technical and organizational issues", pp. 6032 – 6042, in *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS)*, vol. 6, 2000.

[9] D. Draheim, J. Grundy, J. Hosking, C. Lutteroth and G. Weber, "Realistic Load Testing of Web Applications", in *Proceedings of the 10th European Conference on Software Maintenance and Reengineering (CSMR)*, 2006.

[10] D. Krishnamurthy, J. A. Rolia and S. Majumdar, "A Synthetic Workload Generation Technique for Stress Testing Session-Based Systems", pp. 868 – 882, in *IEEE Transactions on Software Engineering,* vol. 32, no. 11, November 2006.

[11] Eclipse. http://www.eclipse.org/.

[12] Developer Resources for Java Technology. http://java.sun.com/.

[13] Yet Another Workflow Language (YAWL). http://www.yawl-system.com/.

[14] Microsoft Web Application Stress Tool. http://www.microsoft.com.

[15] M. Arlitt and T. Jin 2000, "A workload characterization study of the 1998 World Cup Web site", pp. 30 – 70, in *IEEE Network, Special Issue on Web Performance,* vol. 14, no 3, May 2000.

[16] W3C, HTTP/1.1: Status Code Definitions. http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html