University of Ibadan
Journal of Science and Logics in ICT Research

A Review of Two-way Approach to Improve Continuous Responsiveness Provisioning in a Real-Time System

OYEKANMI, E. O.

Achievers University, Owo, Ondo State, Nigeria
e.oyekanmi@achievers.edu.ng

Abstract

A Real-Time (RT) system involves a continual input, process and output of data. This continual operation at a peak level of a server may cause many missed deadlines due to low responsiveness of server in such a real-time system. Continuous responsiveness of an application system means that the system can provide a good quality of service (QoS) and that there is little or no delay in the delivery which can adversely affect the user's experience. In this paper, two distinct webservers' responsiveness were reviewed and two-ways approach namely: job scheduling and admission control were suggested to improve the responsiveness of the underlying hardware used by servers at a peak level.

Keywords: *Responsiveness, Real-Time System, Job Scheduling, Admission Control, WebServer*

1. INTRODUCTION

Responsiveness provisioning is the ability to consistently and effectively making appropriate and timely decisions on customer's requests, short term fluctuations in operating conditions, changes in the overall system environment and how these decisions are executed. A system environment with array of servers could experience fluctuations during processing especially when critical applications are running [1]. This makes servers redundant, because the relevant available information to make the best possible decisions must be routinely applied while still optimizing the underlying hardware component, (like single or dual processor, Random Access Memory (RAM), Hard Drive Disk (HDD), a number of Local Area Network (LAN) cards, Redundant Array of Independent Disk (RAID) controller) or software (like applications and operating system (OS)), on which the functionality of the server depends. In order to maintain continuous responsiveness at peak level, when system is faced with enormous pressure and is to optimize performance and guarantee quality of service, the problem of responsiveness, bottlenecks and process breakdowns in such a system must be addressed to gain competitive advantage.

The world is going online as a result of convid-19 pandemic. Several online lectures are needed to be accessed by various persons from different place. Addressing bottlenecks at peak level of requests on server in a real-time system is the focus of this paper. The objective of this research is to describe the different classification and characteristics of real-time request/task from customers, review the performance of web servers, and provide the mechanisms to improve on the continuous responds to client's real-time tasks from a server.

1.1 Classification of Real-Time Tasks and their Characteristics

Real-time tasks can be categorised into three, based on the way real-time tasks recur over a period of time. These are periodic, sporadic and aperiodic tasks. Periodic task has 4 tuples which include $(\emptyset_i, p_i, e_i, d_i)$. A task with phase $\emptyset_i$ is periodic if it is released for an execution time e_i repeatedly at every p_i seconds which must be completed within a relative deadline d_i time units such that for $e_i \leq \min(p_i, d_i)$ from the time it was issued. Periodic task execute with execution time e_i at regular intervals and

Oyekanmi, E. O. (2021). A Review of Two-way Approach to Improve Continuous Responsiveness Provisioning In a Real-Time System. *University of Ibadan Journal of Science and Logics in ICT Research (UIJSLICTR),* Vol. 6 No. 2, pp. 22-31

can be assigned different priorities with relative deadline d_i in a real-time system. Each of the 4 tuples is greater than zero (0).

1.1.1 Periodic Task

A task is a combination of related jobs. Jobs could be combined in a certain order depending on its type. Jobs that repeat themself after certain period of time are usually periodic in nature. The recurrence time of these set of Jobs is defined by the clock interrupts, hence the task is called a clock-driven task. Periodic task can be represented with 4 tuples as T_i ($\emptyset_i$,p_i, e_i,d_i) where $\emptyset_i$ is the release time of the first job, p_i is the period of the task, e_i is the execution time and d_i is the relative deadline of the task. Each tuple is greater than zero (0). Each task T_i issues periodic requests for $e_i \leq \min(p_i, d_i)$ time units of execution, separated by p_i time units. Every such request, known as job, must complete within d_i time units from the time it was issued. The ratio of the execution cost of T_i to its period, $^{e_i}/_{p_i}$ is defined as utilization [2]. Most of the tasks processed presently in a typical real-time system are periodic in nature. For instance, in chemical engineering, the temperature of a generating-plant, its pressure and chemical concentration are different tasks that normally generated through timer's interrupt [3]. This kind of task is referred to as a static periodic task because it exists from the time of system initialization. However, a periodic task can be dynamically generated during air traffic monitoring as well. This occurs when flight detection flag was raised by the radar at a signal zone.

1.1.2 Sporadic Task

This task repeat itself but not with a fixed period. Three tuples are used instead of four (in periodic task) to represent a sporadic task as T_i (e_i, g_i , d_i). g_i is the minimum separation distance. Task will continue to repeat only when g_i time is over. e_i is the worst case execution time of an instance of the task d_i is the relative deadline of the task. The minimum separation implies that once an instance of a sporadic tasks arise, the next instance cannot occur before g_i time units have elapsed. In other words, any task that occur whose time of occurrence cannot be predicted is a sporadic task. A typical example includes the task that handles fire conditions in a factory and a task that is generated in a robot to handle an obstacle that suddenly appear. As stated in [3], the criticality of sporadic tasks varies from highly critical to moderately critical. An I/O device interrupt or DMA interrupt is moderately critical compared with the fire condition in a factory. The latter is highly critical.

1.1.2 Aperiodic Task

An aperiodic task is same as sporadic. However, the minimum separation (g_i) between two consecutive instances is 0. Also, two or more instances of an aperiodic task might occur at the same time instant. The report in [3] shows that the deadline for an aperiodic tasks can be expressed either averagely or statistically. Aperiodic task are generally soft real-time tasks because it can recur in quick succession leading to bunching of the task instances which might lead to a few deadline misses. Hence, it is very difficult to meet the deadlines of all instances of an aperiodic task. An example is a logging task in a distributed system.

All these tasks need to be attended to accordingly in a robust system. The mechanisms that the underlying hardware of the real-time OS can use to improve continuous responsiveness of the applications at the server's end are discussed in details in the following section.

2. REVIEW OF RELATED WORKS

In a real-time system, the work of a webserver is to respond to clients' requests (majorly files) that reside on the local disk [4] of the system. When a client sends request on network to the server, it is either to fetch a static file (.html, .css, .js) or dynamically generate file using a server-side scripting language. This request is executed on the webserver in a back and forth manner until a result is displayed on the browser. During the back and forth interaction of client and server, decisions are always made based on performance, security and usability. Although the three components are interwoven, performance optimization of server has been the focused components by many researchers in recent years because of its wide usage on daily basis [5, 6]. It was discovered in Prakash, *et. al.* [7] that performance consideration does not only

reduce hardware cost, but also ensure flexibility in hardware and operating system upgrade which has effect on users' experience. In web server, two metrics can be used to measure performance. These include resource utilization and response time.

While resource utilization is based on how busy the resources (central processing unit, local disk and network bandwidth) in the real-time system is, response time defines the delay during TCP connection until the response is received from the point-of-view of a client after submitting a request. Response time, most often, depends on the size of the request submitted to the web-server. Table1 shows a comprehensive review on responsiveness on two web servers using the two measures on Apache and Nginx.

Table 1. Summary of web server performance review-Apache and Nginx [4]

Author	**Web Server**	**Response Time**	**Memory Usage**	**CPU Utilization**
Dreamhost, [8]	Apache	Handles less requests per second at high concurrency	Increase with increase in requests	
	Nginx	More requests per second even at high concurrency	Does not increase with increase in requests	
Jing & Kishor, [9]	Apache	Handles 350-390 requests per second		
Dabkiewicz, [10]	Nginx	Under static files-single worker, it handles 7212 requests per second.		
		With four workers, it can handle 7742 requests per second.		
		Under dynamic files, it handles 1873 requests per second		
	Apache	With single worker, it can handle 7367 requests per second		
		With four workers, it can handle 7242 requests per second.		
		Under dynamic files, it can handle 5142 requests per second.		
Fan & Wang, [11]	Apache	Under dynamic files, it can handle 5142 requests per second.		
	Nginx	Outperforms under all workload concurrency though its throughput decreases under high workload concurrency		
Prakash, Biju, & Kamath, [7]	Apache	5000 requests per second	Increases with increased requests	
	Nginx	Increases by 50% more requests per second than Apache	Increases with increased requests	
He, Karne, Wijesinha, & Emdadi, [12]	Apache	6000 requests per second		At capacity, CPU utilization is 99% and 82% at bare PC server

Peña-Ortiz, et al., [13]	Apache			CPU utilization increases with workload
	Nginx			CPU utilization does not increase with workload

2.1 Initial continuous responsiveness provisioning approach

Andrew and Albert [14] deduced that kernel directs all components' interactions on operating system. Kernel was built on a minicomputer foundation and its major advances including performance, human-computer interfaces and graphics architecture are relatively left untouched years back.

The mechanism for continuous responsiveness according to Andrew and Albert [14] were manipulations on interrupts, device drivers, demand paging and the likes with the notion of not affecting the kernel to prevent rebuilding of another operating system (OS). This persisted because the hardware platform to run a new operating system was not in place. The design only improve quality-of-service (QoS) of servers with low responsiveness.

The fact still remain that users will not quench their thirst for performance increase and functionality of applications as real-time task comes in, therefore expanding or manipulating the number of hardware resources for continuous provisioning is not a viable solution compare with the rate at which requests are increasing [15, 16, 17]. The industry, therefore, has no choice but to improve the efficiency of OS architecture with the hope of meeting customers' growing expectations [15, 18].

3. MECHANISM FOR CONTINUOUS RESPONSIVENESS

The following mechanisms are proposed to be used by operating system to improve the responsiveness of applications at the server's end. The mechanisms are knitted but their mandate are different. These are job scheduling and admission control.

3.1 Job Scheduling

Scheduler aim at one or more of many goals. For example: maximizing throughput (the total amount of work completed per time unit); minimizing wait time (time from work becoming ready until the first point it begins execution); minimizing latency or response time (time from work becoming ready until it is finished in case of batch activity, [19, 20 21] or until the system responds and hands the first output to the user in case of interactive activity), [22].

The work of Job Scheduler (Scheduler) is to enforce measures to prevent traffic violation especially at peak level of workflow. Scheduler manages the sending of different job streams and resolves contention between job streams of different classes at a switching point using a set of queues and other mechanisms like timers which include job arrivals and departures as well as buffer occupancies (queue lengths) so as to satisfy QoS requirements for all classes of job streams' queues. Job scheduler estimator (statistical database) was developed in Jay, *et. al.* [23] and used to control outgoing traffic stream job traffic at the point where jobs are queued.

In Jay [24], a high volume of job information and the high speeds of the flow of job in a real-time system imposes constraints on scheduling decisions which mandated the use of simple algorithm with relatively simple information structures for Job Scheduler.

Every scheduler has a schedulable region denoted by R where R is defined as

$$R = \{j \in \mathbb{N}^n\} \quad (1)$$

Such that scheduler guarantees the QoS for all job classes.

$\mathbb{N}$ is the set of natural numbers, j is the scheduled job of class i with maximum class n

The maximum number $\mathcal{N}^i$ of class *i* allowable into the system from the limits of the schedulable region can be defined by

$$\mathcal{N}^i = \max_{j \in R} j^i \qquad (2)$$

R represents admission policy limits at the scheduling level. This must be enforced by the scheduler to maintain quality of service. Different workflow exist and there are algorithms for scheduling them in a real-time system. The algorithms for the classification of these workflow are discussed in the following section.

3.1.1 Classification of Real-Time Task Scheduling Algorithm

Real-Time task can be classified based on three criteria namely: definition of the scheduling point, task acceptance test and target platform for the scheduler.

The classification by scheduling point depends on three schemes: clock-driven, event-driven and hybrid scheme. In clock-driven schedulers, jobs are executed at a specific time. Time (the interrupts received from a clock) is decided before execution is set. This kind of scheduling is simple and straight-forward. In the event-driven ones, the scheduling points are defined by certain events which precludes clock interrupts, an example is event-driven web servers as stated in Voigt [25]. In hybrid scheme, jobs are to be executed based on time and event-driven.

Some example of algorithms that belong to these groups include Clock Driven (e.g. Table-driven or Cyclic driven); Event Driven (e.g. Simple priority-based, Rate Monotonic Analysis (RMA) or Earliest Deadline First (EDF)); Hybrid (e.g. Round-robin).

The classification based on the task acceptance test can be divided into two categories: Planning-based and Best-effort. In planning-based schedulers, task's dead-lines are first determined before starting the execution. If the deadline can be met and already scheduled tasks will not be interrupted to miss their respective deadlines, then the task is accepted for scheduling. Otherwise, such task is rejected. However, in best effort schedulers, scheduling takes place immediately as task arrives. But no guarantee is given as to whether a task's deadline would be met.

The target platform classification has to do with the platform on which the tasks are to be run. This again can be implemented using three platforms which include: Uniprocessor, multiprocessor and distributed. In Uniprocessor platform, tasks are schedule as it comes. In Multiprocessor platform, decisions are made on any arrived task to determine which processor will execute it, then the task is schedule. It has shared memory and has a global up-to-date state information of all tasks. In Distributed platform, decisions are made on any arrived task to determine which processor will execute it, then the task is schedule. This platform does not have shared memory and there is no global up-to-date state of information of all tasks. The communication among tasks is through message passing and this is costly.

3.2 Admission Control

Admission control algorithms guarantee end-to-end performance by preventing stream overload. Admission control does not only guarantee QoS, but other features which may also be necessary. For example in multimedia server, security measures is needed to validate the user before access permission is granted for the media contents of a video-on-demand server to be viewed. Also, since the data are a marketable commodity, accounting services will be required to charge the users. All these features are the component of a robust admission control algorithm.

The task of the admission controller is to accept or reject arriving called jobs so as to maximize utility function based on the weighted average throughput. The information available to the admission controller includes the boundaries of the schedulable region R specified by the scheduler, the job arrival and departure rates associated with each class of service and the weights used in the utility function [23]. Getting this details in reality might be difficult because, it is the operating system that determines the arrival and departure time. However, an algorithm can be designed and implemented to perform the work of control, thereby increasing the responsiveness of underlying hardware of an operating system.

A server with an admission control scheme will perform optimally when incoming request are at its peak level. The mechanism is based on acceptance or rejection model [26]. A typical example is logistic regression technique.

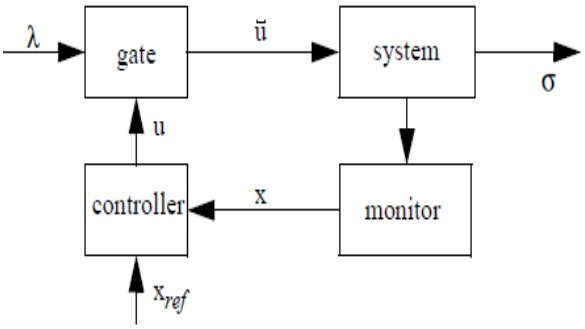

Figure. 1. Admission control structural diagram (Source: http://www.citeseerx.ist.psu.edu/viewdoc/download)

In Figure1, the structural diagram has three sections which include: Gate, Controller and Monitor. x is the control variable which is used by the Controller to capture the proportion of u for which a request could be admitted [27]. This variable is measured by the Monitor. Controller ensures that the variable is in a best proportion to the reference value x_{ref} before admission is granted to the incoming request while Gate rejects the unadmitted requests. The actual admittance rate was defined in equation (3) in term of admittance rate of u and arrival rate of λ as follows:

$$\breve{u}=\min [u, \lambda] \qquad (3)$$

min was used since the admittance rate may never be larger than the arrival rate.

3.2.1 Gate

Example of gate include Percent blocking and Token bucket [26]. Percent blocking mechanism involves the use of fraction or threshold value for requests to be admitted. In Ing-Ray [28], percentage blocking type of gate was used to provide a real-time, continuous service to each client by characterizing their computational requirement using a period T and a computation time C within the period. A thread is then created by the server at the time the client is admitted into the system to invoke periodically, a fraction C/T of the server capacity until the client completes its required service.

In the case of token bucket, token comes out at a definite rate. An arriving request is therefore admitted if there is a token available for it. Token bucket, as stated in Wikipedia [29], can be conceptually understood as follows:

i. A token is added to the bucket every $1/r$ seconds, where r is the average rate.
ii. The bucket can hold at the most b tokens, where b is the token depth. If a token arrives when the bucket is full, it is discarded.
iii. When a packet (network layer Protocol Data Unit) of n bytes arrives,
   - if at least n tokens are in the bucket, n tokens are removed from the bucket, and the packet is sent to the network.
   - if fewer than n tokens are available, no tokens are removed from the bucket, and the packet is considered to be non-conformant. n is the maximum size of the bucket.

Token bucket is used to ensure that the incoming packet has sufficient tokens before admitting into the network for processing as shown in Figure 2. Token rate regulates transfer of packets and saves permission to send large bursts. This means that the bursts of up to n packets can be sent at once thereby allowing burstiness in the output stream to be regulated and gives faster response to sudden bursts of input.

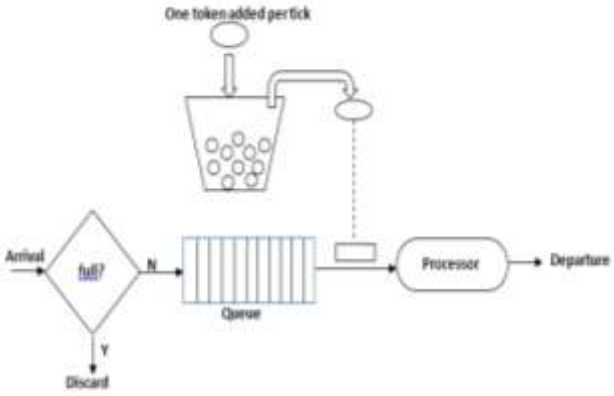

Figure 2: Token-bucket algorithm sketch

Gate uses a dynamic measure (window size) to ensure that the number of requests to be processed or waiting in the system does not exceeds the upper boundary. There could be

alternation of window size as the traffic conditions changes.

3.2.2 Controllers

An admission control mechanism utilizes various controllers. Some of the most common controllers are the Static controller, the Step controller and the Proportional Integrating Derivative (PID), PID-controller [26]. A static controller uses a fixed acceptance rate, u_{fix} which is set so that the average value $\check{x}$ of the control variable should be equal to the reference value ρ_{ref}. In this case, u_{fix} is given by

$$u_{fix} = \frac{\rho_{ref}}{\check{x}} \qquad (4)$$

The main objective in Step controller as stated in [26] is to keep the control variable between an upper and a lower level. Usually, in this case, the value of the variable is inversely proportional to the admittance rate. The control law is as follows:

$$u(t+1) = \begin{cases} u(t) - s & y(t) > y_{ref} + \varepsilon \\ u(t) + s & y(t) > y_{ref} - \varepsilon \end{cases} \qquad (5)$$

where the value of s decides how much the rate is increased/decreased and the value of ε decides how much the control variable may deviate from the reference value.

The PID-controller uses three actions: one proportional, one integrating, and one derivative. The control law in continuous time is as follows:

$$u(t) = Ke(t) + \frac{K}{T_i}\int_0^t e(v)dv + KT_d\frac{d}{dt}e(t) \qquad (6)$$

where $e(t)$ is the error between the control variable and the reference value, that is

$$e(t) = y_{ref} - y\,(t). \qquad (7)$$

The gain K, the integral time T_i, and the derivative time T_d are the controller parameters that are set so that the controlled system behaves as desired.

A large value of K makes the controller faster, but weakens the stability. The integrating action eliminates stationary errors, but may also make the system less stable. The derivative action improves the stability, however, in a system with a bursty arrival process the derivative action may cause problems. Therefore, the derivative action is usually either deleted (i.e. $T_d = 0$) or low pass filtered to remove the high frequencies.

3.2.3 Monitor

The Monitor oversees server utilization on a continuous control interval. During the last control interval, fraction of time of an idle process is calculated. The result is subtracted from one and this makes the server utilization value [26]. The Central Processing Unit (CPU) is not needed when a process is idle. The priority level of an idle process is usually set to the lowest possible value as a quantizing measure in server utilization. With this approach, the operating system, where the admission control mechanism runs, performs certain time resolution as regards function calls during this process. This means that there has to be a logical control interval for the smooth running of the process. It has to be long enough not to be affected by the time resolution effects, and short enough so that the controller responds quickly.

A typical example where admission control is needed is in a multimedia real-time server. The innumerable quest for continuous multimedia today has contributed to users' expectancy in the area like high-quality multimedia, snappy operation and interactive applications. The responsive user interfaces and stringent real-time guarantees from the systems that host the resources is of great importance. However, there are components that make up a multimedia server that does the real-time streaming, these are explained in the next section

3.2.4 Architecture of Admission Control in a Media Server

Media server has some certain components which are shown in Figure 3. These include Platform Manager (PM), Admission Control Unit (ACU), Resource Manager (RM) and Monitor. The front end of media server is the PM, with the help of which user sends a request for a video file. ACU accepts request from PM and check for the availability of the resources that will guarantee the continuous responsiveness of the requested file from server. RM records available resources and its update and provides ACU with it when needed. After a machine is assigned to a user, the Monitor maintains the state of the machine and responds back to the RM. The media server components are explained in detail as follows.

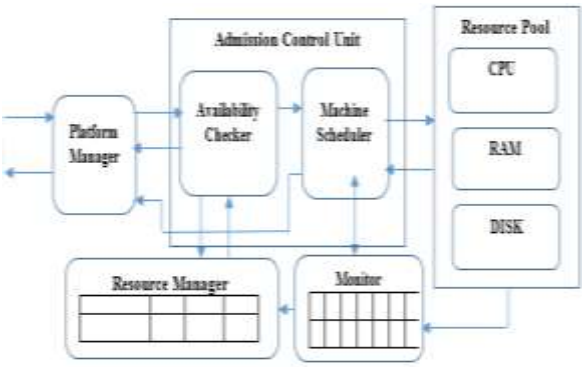

Figure 3: Media Server Architecture

a) Platform Manager (PM)

Platform Manager (PM) acts as an interface between the user and the system. It provides user with the ability to initiate a request and then define the user's specific resource requirement which is divided into two parts. One is the Host requirement and other is Environment requirement.

The host requirement part consists the type of processor, number of Central Processing Unit (CPU) cores, Random Access Memory (RAM) size and Internet Protocol (IP) address. The environment requirement part consist the type of Operating System (OS), runtime environment and so on. Once these information are obtained, PM communicates Admission Control Unit (ACU) in a bidirectional way. It is the interface through which users communicate the media server. Communications that involve scaling or stopping the machine is also done via this interface.

b) Admission Control Unit

The Admission Control Unit (ACU) is responsible for managing the user's requests and sending instruction to other units for provisioning of a platform. As the request comes from the PM, the Availability Checker passes the information to Resource Manager (RM) and asks to check for the user's required resources. The required resources and environment are sent to the ACU by the RM if the duo are available and compatible with the users system. This checked information are then forwarded by the Availability Checker Unit to the Machine Scheduler. Machine Scheduler is responsible for dispatching the job to the best-fit resource to process it. It notifies the PM of the successful allocated resource's status and machine ID. It also informs the Monitor so that it makes an entry in its table to keep track of running systems.

c) Resource Manager

Resource Manager (RM) archives information of all available machines and their runtime environment. The resource information stored in a table-like form include RAM size, CPU capacity and Disk size. Requests from ACU about any of these resources are checked from RM to ensure that virtualization does not occur because of their limited number. In other words, RM matches user's request with the available resources to guarantee continuous responsiveness provisioning.

d) Monitor

The essence of monitor component is to keep track of the state of each machine computation till the point of being available for another user. It updates the free list of machines maintained by the RM.

Continuous responsiveness majorly depends on the underlying hardware, however software approach could be used to make the work done. With consideration of the two mechanisms: Request/Job scheduling and Admission Control, the responsiveness of the underlying hardware in a server would be improved.

4. CONCLUSION

The objective of resource provisioning is to detect and provide the appropriate resources to the suitable workloads on time, so that applications can utilize the resources effectively. However, it is constantly complex to make selection for an appropriate approach that could provide continuous responsiveness for applications system. Distinctly, this paper has discussed Job scheduling and Admission Control as two major mechanisms that could improve continuous responsiveness provisioning in a real-time application system. Admission control schemes does not have mathematical model in queuing theory However, with control theory, analysing queuing systems will be possible to model using control theoretic methods. With this, a good admission control mechanisms can be designed for a real-time systems.

References

[1] Amies, A., Sanchez, J., Vernier, D., Zheng X. D. (2011). Monitor services in the cloud IBM developer Works.

[2] UmaMaheswari C. D. (2003). An Improved Schedulability Test for Uniprocessor Periodic Task Systems. In Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03), IEEE computer Society

[3] Kharagpur (2019). Real-Time Task Scheduling.Version 2 CSE IIT,URL:http://www.nptel.ac.in/courses/Webcourse-contents/IIT%20Kharagpur/Real%20time%20system/pdf/Module2.pdf

[4] Douglas, K., Sipiwe, C., Muwanei, S. (2017). Web Server Performance of Apache and Nginx: A Systematic Literature Review. Computer Engineering and Intelligent Systems, 8(2).

[5] Xianghua, X., Tingting, X., Yuyu-Yin, J. W. (2013). Performance evaluation model of Web servers based on response time. IEEE Conference Anthology, 1-5

[6] Zhaoyang, Q.,Wei-Wang, Z. L. (2010). Web Server Optimization Model Based on performance analysis. IEEE, 1-4.

[7] Prakash, P., Biju, R., Kamath, M. (2015). Performance Analysis of Process Driven and Event Driven Web Servers. IEEE Sponsored 9th International Conference on Intelligent Systems and Control (ISCO) , 1-7

[8] Dreamhost [online] http://www.dreamhost.com (2016)

[9] Jing, Z., Kishor, S. T. (2011). Performance Modeling of Apache Web Server Affected by Aging. Third International Workshop on Software Aging and Rejuvenation, 1-6

[10] Dabkiewicz, S. (2012). Web Server Performance Analysis. Lia project, 1-14

[11] Fan, Q., Wang (2015). Performance Comparison of Web Servers with Different Architectures: A Case Study using High Concurrency Workload. Third IEEE Workshop on Hot Topics in Web Systems and Technologies, 37-42

[12] He, L., Karne, R., Wijesinha, A., Emdadi, A. (2009). A Study of Bare PC Web Server Performance for Workloads with Dynamic and Static Content. 11th IEEE International Conference on High Performance Computing and Communications, 1-6

[13] Peña-Ortiz, R., Gil, J., Sahuquillo, J., Pont A. (2012). Analysing web server performance under dynamic user workloads. Computer communications, vol 36, No. 4, pp. 386-395

[14] Andrew, S. T., Albert, S. W. (2005). Operating Systems Design and Implementation (3rd Edition). Upper Saddle River, NJ, USA: Prentice-Hall, Inc

[15] Christos, K. (2013). Resource efficient computing for warehouse-scale datacenters. In: Design, Automation Test in Europe Conference Exhibition, 1351–1356

[16] Kushagra, V. (2010). Datacenter Power Efficiency: Separating Fact from Fiction. Invited talk at the 2010 Workshop on Power Aware Computing and System

[17] Mark, H. (2005). Scaling, power, and the future of CMOS. In: Electron Devices Meeting, IEDM Technical Digest. IEEE International.

[18] Luiz, A. B. (2011). Warehouse-Scale Computing: Entering the Teenage Decade. In: Proceedings of the 38th Annual International Symposium on Computer Architecture. ISCA '11. San Jose, California, USA: ACM

[19] Liu, C., James W., Layland (1973). Scheduling Algorithms for Multiprogramming in a Hard- Real-Time Environment. Journal of the ACM. ACM. 20 (1): 46–61

[20] Kleinrock, L. (1976). Queueing Systems, Computer Applications, Wiley-Interscience, 2(1) , p. 171

[21] Feitelson, Dror G. (2015). Workload Modeling for Computer Systems Performance Evaluation. Cambridge University Press. Section 8.4 (Page 422)

[22] Silberschatz, A., Galvin, P. B., Gagne, G. (2012). Operating System Concepts (9ed.). Wiley Publishing. p. 187.

[23] Jay, M. H., Aurel, A. L.,Giovanni P. (1993). A Separation Principle between Scheduling and Admission Control for Broadband Switching. IEEE J.Select. Areas Community, Vol. 11, No 4, pp. 605-615

[24] Jay, M. H., Aurel A. L., Giovanni P. (1991). Real-time scheduling with quality of service constraints. IEEE J.Select. Areas Community, Vol. 9, pp. 1052-1063

[25] Voigt, T. (2002). Overload behaviour and protection of event-driven web servers, In Proceedings of the International Workshop on Web Engineering, May 2002, Pisa, Italy

[26] Mikael, A., Maria, K., Anders, R. (2019). Modelling and Design of Admission Control Mechanisms for Web Servers using Non-linear Control Theory. Lund Institute of Technology Box 118, SE-221 00 Lund, Sweden

[27] Kihl, M. (1999). Overload control strategies for distributed communication networks, Ph.D thesis, Dep. of Communication Systems, Lund Institute of Technology, Sweden

[28] Ing-Ray C. Chi-Ming C. (1996). Threshold-Based Admission Control Policies for Multimedia Servers. Computer Journal.

[29] Wikipedia contributors. (2019, June 5). Token bucket. In Wikipedia, The Free Encyclopedia. Retrieved 08:39, September 6, 2019, from https://en.wikipedia.org/w/index.php?title=Token_bucket&oldid=900385526