EFFECTS OF SYSTEM PAGING / THRASHING OVERHEAD ON SYSTEM PERFORMANCE BASED ON DEGREE OF MULTIPROGRAMMING

Akhigbe–Mudu Thursday Ehis

Department of Computer Science, Federal University of Agriculture Abeokuta, Nigeria

ABSTRACT: There is usually a strong concern about the functionality of computer systems and very little concern about its performance. In general, performance becomes an issue only when the performance becomes perceived as being poor by users. Major constraints on systems show themselves in form of external symptoms, stress conditions and paging being the chief forms. Thrashing occurs when a computer's virtual memory subsystem is in a constant state of paging, rapidly exchanging data in memory for data on disk, to the exclusion of most application processing. This causes the performance of the system to degrade by multiple orders of magnitude. We developed analytical models with computational algorithms that provide the values of desired performance measures. Lyaponuv stability criteria provide sufficient conditions for the system validity, the result shows that the savings generated from the page- based, offsets the additional time needed to process the increase number of page- faults.

KEYWORDS: Paging, Thrashing, Multiprogramming, Interfault, Frames and overhead

1. INTRODUCTION

Paging from the layman point of view is a technique used by virtual memory operating system to help ensure that the data you need is available as quickly as possible. When a program needs a page that is not in the main memory, the operating system copies the required page into memory and copies another page back to the disk. As this process continues, a page fault occurs when the address of the page being requested is invalid and in that case, the application is usually aborted. It is the page – in rate that is of primary concern, because page – in activity occurs synchronously [MAD94]. A page – in incurs a time cost for the physical I/O and a more significant increase in processor usage. Page – in activity slows down the rate at which transactions flow through the system, that is, transactions take longer to get through the system. If a system does not have enough pages, thrashing is a high paging activity, and the page fault rate is high. This leads to CPU utilization. In modern computers, thrashing may occur in the paging system (if there is not sufficient physical memory or the disk access time is overly long), or in the communications system (especially in conflicts over internal bus access), etc. Depending on the configuration and algorithms involved, the throughput and latency of a system may degrade by multiple orders of magnitude as a result of excessive paging [MA82]. In virtual memory system, thrashing may be caused by programs or work load that present insufficient locality of reference, if the working set of a program or a workload cannot be efficiently held within physical memory, then constant data swapping, i.e. thrashing may occur. Hence, the performance evaluation of computer system and networks has thus become a permanent concern of all who use them.

1.1. Identifying System Constraints

Major constraints on system show themselves in the form of external symptoms, stress conditions and paging being the chief forms. The fundamental thing that has to be understood is that practically every symptoms of poor performance arises in a system that is congested. For example, if there is a slowdown in a system, transactions doing data set activity pile up; there are waits on strings; there are more transactions in the system, there is therefore a greater virtual storage demands, there is paging, and because there are more transactions in the system, the task dispatcher uses more processor power scanning the task chains. We then have task constraints; transaction class limit is exceeded and adds to the processor overhead because of retries and so on. The result of this is that the system shows heavy use of its resources and this is the typical system stress. Thrashing occurs when a computer's virtual memory subsystem is in a constant state of paging, rapidly exchanging data in memory for data on disk, to the exclusion of most application level processing. This causes the performance of the system to degrade or collapse. Hence, the battle cry for system optimal performance resonates with a growing and wide spread concern to improve the performance of computer systems and to fail to do so today will amount to inviting catastrophic system failure tomorrow.

1.2. Memory Queuing

Multiprogramming is a central part of modern operating systems. It allows for several programs (jobs), transactions, processes and requests to be memory resident at the same time, sharing the system resources, such as processors and I/O devices. Most

programs do not overlap their I/O and processing activities, so that while one program is making use of the processor, others are undergoing I/O processing. Therefore, multiprogramming leads to a better use of the system resources, by keeping I/O devices running concurrently with processors. However, due to the finitude of real memory, the number of programs in execution may affect system performance. For example, in virtual memory systems, as the multiprogramming level increases, the paging and swapping activities also increase and may cause a reduction of throughput of productive work. Thus, there must be control of the maximum number of programs allowed to share memory at the same time. Usually, operating systems control the overall load by acting on the degree of multiprogramming.

1.3. Problem Descriptions

Paging moves individual pages of processing from main memory to disk and back, as a result generates I/O traffic. Each time a page is needed that is not currently in memory, a page fault occurs. An invalid page fault occurs when the address of the page being requested is invalid; in this case, the application is usually aborted and depending on the configuration and algorithms involved, the throughput and latency of a system may degrade by multiple orders of magnitude. The intensity of paging activities is measured by page – ins (pages moved from disk to memory), and page – outs (pages moved from memory to disk), as exhibited in figure 1. Page – ins incurs a time cost for physical I/O and a more significant increase in processor usage, since it has to wait until the page transfer completes. As a result, most present day operating systems use demands paging, which means that a page of code or data is not read into memory until the page is needed. In an over committed situation, where the number of pages actively in use exceeds the number of pages of physical memory, it becomes necessary to select pages that will be removed from physical memory in order to make room for new pages. When the paging activity starts to increase, the load on the paging device increases and it eventually becomes the bottleneck. The throughput decreases and eventually goes to zero (Thrashing) and as such, paging activity should be included in performance model systems. The rest of this paper is structured in the following manner: - section (2) discusses the literature review, section (3) focuses on the methodology of the work, while section (4) analyzes the paging activities, implemented the method discussed in the body of the paper and evaluated the result using Lyapunov stability criteria. Section (5) discusses the summary and conclusion of the paper.

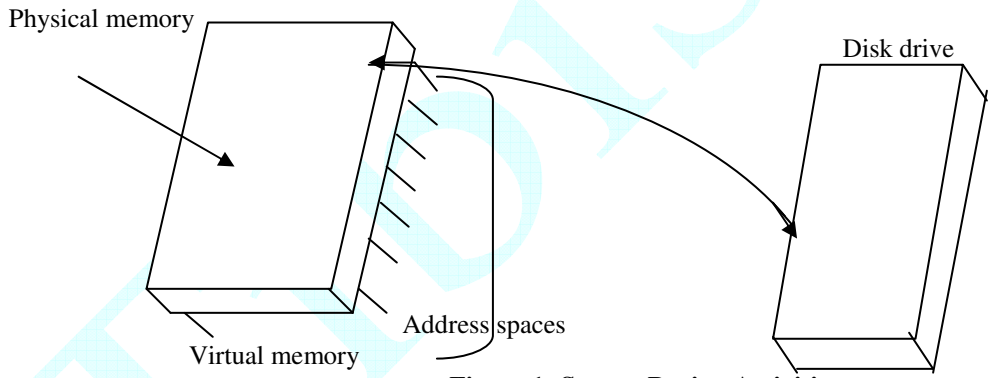

Figure 1: System Paging Activities

2. LITERATURE REVIEW

Poor performance of the operating systems can have considerable impact on application performance. To avoid the operating system from limiting application performance, it must be highly current. In the early 1990's, researchers identified memory performance as critical to system performance. [KGP89] noted that Cache performance and Coherency are critical aspects of hardware which must be taken into account. They explicitly advocated that operating systems must be optimized to meet the demands of users for high performance. They pointed out that OS are large and complex and the optimization task is difficult and without care, tuning can result in increased complexity with little impact to the end – user's performance. The drawback here was the key to focus optimization by identifying performance problems. [WW00] implementation and analysis of page based compression in the OS/2 operating system yielded two results with respect to memory. First, OS with page based compression uses less virtual memory than an operating system without page – based compression. In an unconstrained environment, the system with page based compression will generate fewer page faults than the system without page based compression, which translates into faster performance. Secondly, a system with page- based compression requires more locked (wired) memory than a system without page based,

which effectively reduces the amount of memory available to the rest of the system. This will produce no notable effect in an unconstrained environment; however, a small reduction in the amount of memory will cause an increase in the number of page faults. Also we found that the time saved due to compression offsets the additional time caused by the increase in number of page faults. Though page based compression improves the performance of a demand paging operating system, it reduces the amount of memory available to the rest of the system which does not support our effort. [BD06] used the number of queries being executed concurrently as a measure of multiprogramming level, and choose to define multiprogramming level as being the number of queries in any phase of execution. The drawback here is that data sharing is based on observation that in certain data – based applications, multiple queries may be accessing the same relations concurrently, and there is a high probability that index pages will be accessed by these applications. [AEP11] proposed a tool for planning and control, costing systems play a considerable role in providing information needs for managers. This research seeks to explore such organizational factors as organization size, industry type, cost structure, the importance of cost information and products and services. This study fails to incorporate other important variables which are likely to influence cost system design.

2.1. System thrashing overhead

Overhead denotes hardware resources use by the operating system. It can be viewed as composed of two parts:- a constant one and a variable one. The former corresponds to those activities performed by an OS that do not depend on the level of system load, such as the CPU time handle an I/O interrupts. The variable component of overhead stems from activities that are closely associated with the system load [ZNQ93]. For example, as the number of jobs in memory increases, the work done by memory management modules also increases (Thrashing) [MAD94]. Basically, there are two approaches for representing overhead in performance models. One approach uses a special class of the model for representing the overhead of the OS activities performed on behalf of the application programs. There are problems associated with this approach. Because of its variable component, the service demands of the special class have to be made load – dependent. Thus, whenever the intensity parameters (e.g. multiprogramming level and arrival rate) of the application classes change, the service demands of the overhead class also have to be modified. The inter-dependency between overhead parameters and multiprogramming mix may make this approach impractical [MLT12].

3. METHODOLOGY

Most modern computer systems have paging disks that hold the pages of the virtual addresses of the several processes. As a process page fault rate (number of page faults divided by the number of memory references) increases, a greater portion of the process execution time is spent on the paging device. The load on the paging device affects all processes in execution. We shall show how the paging activity can be taken into account when modeling computer system.

3.1. Modeling Paging Activity

Let pag be the index of the paging device but this device can also be used to store files other than process address spaces, a process service demand at the paging device may be decomposed into two components: one due to paging $\left(D^{P}{}_{pag}\right)$ and another not due to paging $\left(D^{NP}{}_{pag}\right)$ which is a given parameter.

$$D_{pag} = D^{NP}{}_{pag} + D^{P}{}_{pag} \quad (1)$$

The problem here is how to estimate $\left(D^{P}{}_{pag}\right)$. We can say that average service demand at the paging device due to paging is equal to the average number of page faults multiplied by the average disk time to service a page fault (S_{pag}).

Hence

$$D^{P}{}_{pag} = average.number.of.page.faults \chi S_{pag} \quad (2)$$

Since the average service time per page fault can easily be computed as a function of the disk physical characteristics and the size of a virtual page, our problem reduces to estimating the average number of page faults per process. Let f be the number of page frames allocated to a process in maim memory. Let $IFT(f)$ be a function that gives the inter–fault time of a process when f page frames are allocated to the process. The inter–fault time is defined as the average time between consecutive page faults. So, if we divide the total CPU time of a process $\left(D_{cpu}\right)$ by the value of the $IFT(f)$ function, we obtain the average number of page faults. The graph of the IFT function is monotonically increasing as shown in figure 2.

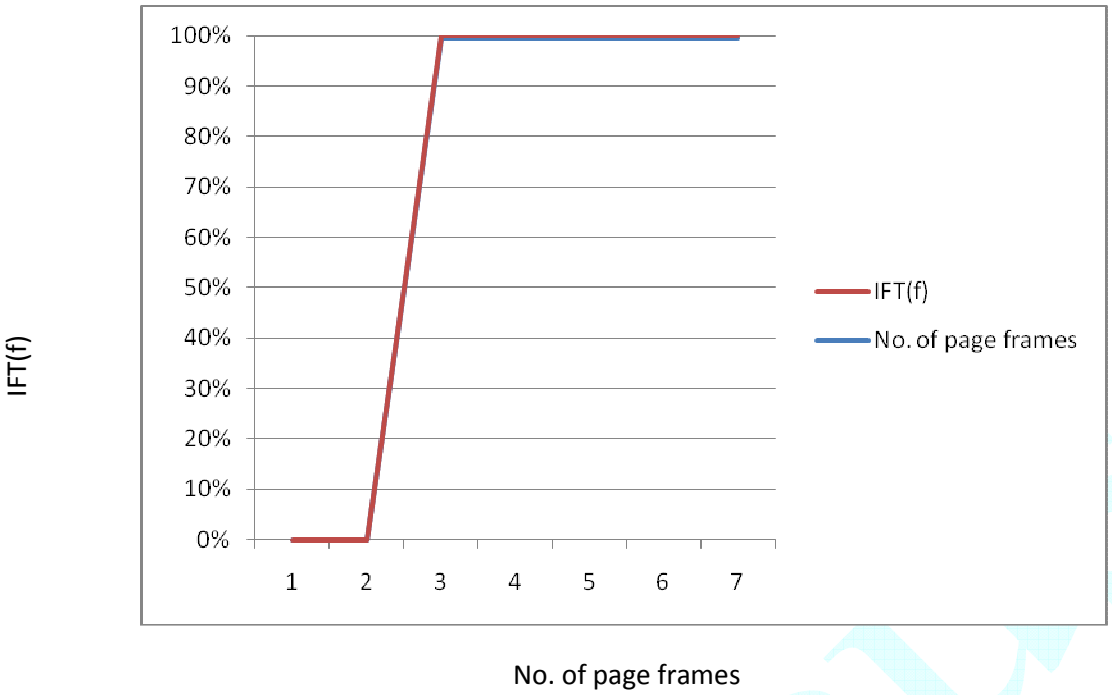

Figure 2: Interfault Time versus Number of Page Frames

Initially, the interfault time grows very fast as the process acquires more page frames. However, as the number of page frames approaches the size of the process working set, the rate of increase of the interfault time decreases sharply and tends to zero as the number of page frames allocated to the process approaches its total number of pages. At this point, there are no page faults and the interfault time becomes equal to the total CPU time for the process. A possible expression for such a function is:

$$IFT(f) = \frac{D_{cpu}}{1+(a/f)^2-(a/F)^2} \quad (3)$$

Where F is the number of virtual pages that compose a process address space and the constant "a" may be obtained from measurement data.

Let NP be the total number of page frames to be divided by all processes and let n be the multiprogramming level. Assuming that the page frames are equally divided among all processes, each will be allocated a number of frames f given by NP/n. Hence, the service demand due to paging can be written as:

$$D^{p}{}_{pag} = \frac{D_{cpu}}{IFT(NP/n)} xS_{pag} \quad (4)$$

Using Eq. (3) in Eq. (4), we get that:

$$D^{p}{}_{pag} = [1+\left(\frac{a}{NP/n}\right)^2-\left(\frac{a}{F}\right)^2]xS_{pag} \quad (5)$$

Therefore, using Eq. (5) in Eq. (1), we get that:

$$D_{pag} = D^{NP}{}_{pag} + \left[1+\left(\frac{a}{NP/n}\right)^2-\left(\frac{a}{F}\right)^2\right]xS_{pag} \quad (6)$$

Note that Eq. (6) is valid only when the degree of multiprogramming is such that there are not enough page frames to store the complete address space of all processes (i.e. when $nxF > NP$). Otherwise, there are no page faults and the paging device service demands due to paging activity are zero. We can now use Eq. (6) to adjust the residence time recurrence expression for the paging device in the algorithm given in figure (3).

$$R'_{pag}(n) = \quad (7)$$

$$\begin{cases} \left[D^{NP}{}_{pag}+\left(1+\left(\frac{a}{NP/n}\right)^2-\left(\frac{a}{F}\right)^2\right)xS_{pag}\right]x\left[1+\bar{n}_{pag}(n-1)\right] \\ D^{NP}{}_{pag} \end{cases}$$

3.2. Throughput Estimation

We see that to compute the residence time $R'_{i,r}(\bar{N})$ we need the values of the probabilities $P_i(j-1/\bar{N})$. To compute these probabilities we need the values of the throughputs $X_{o,r}(\bar{N})$, which depend on the residence time values. We propose the following iterations approach:

1. Estimate initial values for the throughputs $X_{o,r}(\bar{N})$. These estimates can be obtained by approximating the throughput by its asymptotic upper bound, namely $X_{o,r} \le \min\left\{N_r / \sum_{i=1}^{k} D_{i,r}, \frac{1}{\max_i}(D_{i,r})\right\}$. Although this is rather loose upper bound, it is usually good enough as a starting point for the iteration,
2. Compute the probabilities $P_i(j-1/\bar{N})$
3. Compute the residence times,
4. Compute new values for the throughputs using Little's result as $X_{o,r}(\bar{N}) = \frac{N_r}{\sum_{i=1}^{k} R'_{i,r}(\bar{N})}$
5. If the relative error between the throughputs obtained in the current iteration and the previous one is greater than a certain tolerance $\in$ then goto step (2). Otherwise, compute the final metrics and stop.
   This approach is specified in details in the form of algorithms in figure (3).

3.3. Algorithms

Input parameters:

$D_{i,r}, S, \bar{N}, k, R, \in, \alpha_i(j)$

initialization:

for.r := 1.*to.R.do*

for.i := 1.*to.k.do.if*. $D_{i,r} > 0$.*then*

$\bar{m}_{i,r}(\bar{N}) = N_r / K_{r,i}$

$$for.r := 1.to.R.do.X_{o,r}^{\ prev} = \min\left\{N_r / \sum_{i=1}^{k} D_{i,r}, \frac{1}{\max_i}(D_{i,r})\right\}$$

$Error = \in +1; \{force.entering.loop.for.first.time\}$

Iteration loop

while.error > $\in$.*do*

begin

compute.Queue.lenghts.for.Non – LD.devices

for.all.non – LD.device.i.do.forr := 1.*to.r.do*

$$\bar{n}(\bar{N} \_ \bar{I}_r) = \frac{N-1}{N_r}\bar{n}_{i,r}(\bar{N}) + \sum_{i=1}^{k}\bar{n}_{i,j}(\bar{N})$$

compute.probabilies.for.LD.devices

$$\begin{cases} P_i(0/\bar{N}) + \left[1 + \sum_{j=1}^{\bar{N}} \prod_{k=1}^{j} \sum_{i=1}^{R} D_{i,r} X_{o,r}^{\ prev}(\bar{N}-1)\right]^{-1} \\ P_i(j/\bar{N}) = P_i(o/\bar{N}) \prod_{k=1}^{j} \frac{\sum_{r=1}^{R} D_{i,r} X_{o,r}^{\ prev}(\bar{N})}{\alpha_i(k)} \\ j = 1..., |\bar{N}| \end{cases}$$

*compute.*Re*sidence.Times*

for.i := 1.*to.k.do*

for.r := 1.*to.R.do*

$$R_{i,r}'(\bar{N}) = \begin{cases} D_{i,r}\left[1 + \bar{n}_j(\bar{N} - I_r)\right] \\ D_{i,r} \\ D_{i,r} \sum_{j=1}^{|\bar{N}|}\left[j/\alpha_i(j)\right] P_i(j-1/\bar{N}) \end{cases}$$

computethroughputs

$$for.r := 1.to.R.do.X_{o,r}^{\ curent}(\bar{N}) = \frac{N_r}{\sum_{i=1}^{k} R_{i,r}'}$$

compute.queue.lenghts.per.class

for.all.non – LD.device.j.do.for.r := 1.*toR.do*

$\bar{n}_{i,r}(\bar{N}) = X_{o,r}^{\ curr}(\bar{N}) x R'_{i,r}(N)$

compute.relative.error

$$Error := \max\left|\frac{X_{i,r}^{\ curr}(\bar{N}) - X_{o,r}^{\ prev}(\bar{N})}{X_{o,r}^{\ curr}(\bar{N})}\right|$$

Prepare for next iteration

$$for.r := 1.to.R.do.X_{o,r}^{\ prev}\left(\bar{N} := X_{o,r}^{\ curr}(\bar{N})\right)$$

$End; \{while\}$

Figure 3: Multiple Class Algorithms

4. IMPLEMENTATION

Example: A computer system has one CPU and two disks, $((disks: D_1.and.disk: D_2), disk.D_2$ is used for paging purposes only. The service demands at the CPU and at disk D_1 are 0.02 and 0.06 seconds respectively. The paging disk has an average service time equal to 30msecs. The system has 10Mbytes of memory available for paging. The size of the page frame is equal to 1024 bytes. The address space of each process is equal to 2 Mbytes. The constant "a" for the interfault time function is assumed to be equal to 4000. Plot a graph of the throughput versus the multiprogramming level.

We need to compute the mode parameters from the problem data. The number of virtual pages per address space F, is equal to $2x1024x1024/1024 = 2048.pages.$

The total number of page frames is equal to $NP = 10x1024x1024/1024 = 10,240.pages$

Also $D_2^{NP} = 0$ since the paging disk is used for paging only and $S_{pag} = 0.03\text{sec}$. We

Can now use the Algorithms and compute the throughput for various values of the degree of multiprogramming n. we need to use Eq. (7) for the residence time of the paging disk. For the other devices we use the normal residence time equation, In this example, the residence time equation becomes:

$$R'_{D_2}(n) = \left[1+\left(\frac{4000}{10,240}n\right)^2 - \left(\frac{4000}{2048}\right)^2\right] x0.03x\left[1+\bar{n}_{pag}(n-1)\right]$$

$$= \left(1+0.153n^2 - 3.81\right).x0.03x\left[1+\bar{n}_{pag}(n-1)\right]$$

$$for.n > 5 = 10,240/2048$$

$$for.n \le 5$$

$$n = 1,...,N$$

$$R'_{D_2}(n) = 0$$

The resulting throughput graph is given in figure 4. It predicts the expected result, namely that at the beginning, the throughput increases with the degree of multiprogramming. After the paging activity starts to increase, the load on the paging device increases and eventually becomes to the bottleneck. The throughput decreases and eventually goes to zero. This phenomenon is called thrashing.

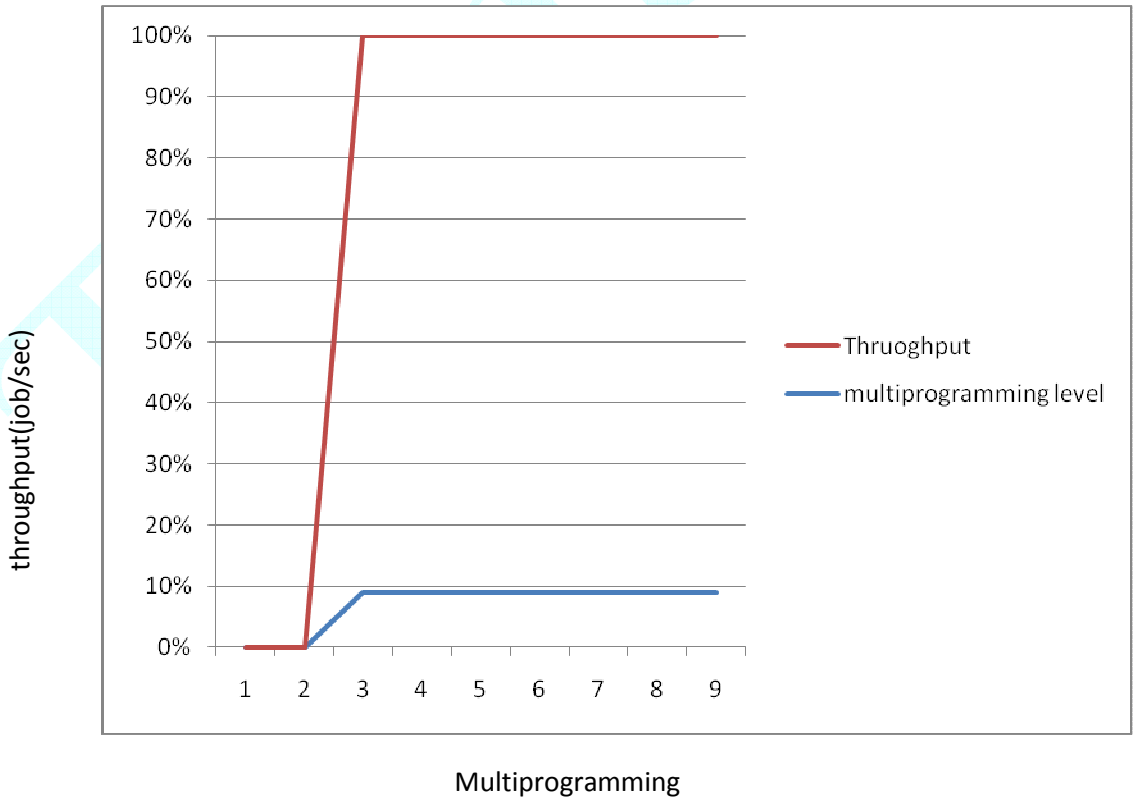

Figure 4: Throughput versus Multiprogramming Level

4.1. System Evaluation

For a given system, stability is usually the most important thing to be determined. Lyaponuv stability criteria is the most general method for the determination of stability of non- linear / or time – varying systems [B+06]. Lyaponuv presented two methods (first and second methods) for determining the stability of dynamic systems described by ordinary differential equations. The first method consists of all procedures in which the explicit form of the solutions of the differential equations is used for the analysis. The second method does not require the solutions of the differential equations. Therefore, the second method is quite convenient for the stability analysis of non – linear systems, for which the exact solution may not be obtained because solving non-linear / or time- varying state equations is usually very difficult. Lyaponuv provides a sufficient condition for asymptotic stability and this theorem is stated as follows;

4.2. Theorem

Suppose that a system is described by

$$\dot{x} = f(x,t)$$

$$where \quad f(0,t) = 0 \quad for.all.t$$

If there exist a scalar function $v(x,t)$ having continuous, first partial derivatives and satisfying the following conditions:

1. $v(x,t)$ is positive definite,
2. $\dot{v}(x,t)$ is negative definite,

Then the equilibrium state at the origin is uniformly asymptotically stable.

We shall consider the second order system described in the paper by:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}0.1\\ -1.-1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$ and we shall determine the stability of this state.

Let us assume a tentative Lyaponuv function, $v(x) = x'Px$

$$where.P.is.to.be.\det er\min ed.from: A'P + PA = -1$$

$$or$$

$$\begin{bmatrix}0.-1\\ 1.-1\end{bmatrix}\begin{bmatrix}p_{11}.p_{12}\\ p_{21}.p_{22}\end{bmatrix} + \begin{bmatrix}p_{11}.p_{12}\\ p_{21.}p_{22}\end{bmatrix}\begin{bmatrix}0.1\\ -1-1\end{bmatrix} = \begin{bmatrix}-1.0\\ 0-1\end{bmatrix}$$

By expanding this matrix equation, we obtain three simultaneous equations as follows:

$$-2p_{12} = -1$$

$$p_{11} - p_{12} - p_{22} = 0$$

$$2p_{12} - 2p_2 = -1$$

$$solving.for.p_{11}, p_{12}, p_{22}.we.obtain$$

$$\begin{bmatrix}p_{11.}p_{12}\\ p_{21}.p_{22}\end{bmatrix} = \begin{bmatrix}\frac{3}{2}.\frac{1}{2}\\ \frac{1}{2}.1\end{bmatrix}$$

To test the positive definiteness of P, check the determinants of the successive principal minors.

$Q = -\left(A^*P + PA\right)$ is positive definite

$$Q_1 = \frac{3}{2} > 0$$

Clearly $$\begin{bmatrix}p_{11}.p_{12}\\ p_{21}.p_{22}\end{bmatrix} = \begin{bmatrix}\frac{3}{2}.\frac{1}{2}\\ \frac{1}{2}.1\end{bmatrix} > 0$$

Clearly, P is positive definite. Hence the equilibrium state at the origin is asymptotically stable in the large and the Lyaponuv function is:

$$v = x'p_x = \frac{1}{2}\left(3x_1^2 + 2x_1x_2 + 2x_2^2\right)$$

$$\dot{v} = -\left(x_1^2 + x^2_2\right)$$

If the page fault rate gets below the computed range, then the process losses a frame and if the page fault rate gets too high, the process gains a frame (see figure 5), and this results in fewer page – faults. Therefore, our approach has been able to evaluate the effectiveness of the analytical model which is composed of a set of formulas and / or computational algorithms. Interestingly, we find that the savings generated from page- based offsets the additional time needed to process the increase number of page - faults.

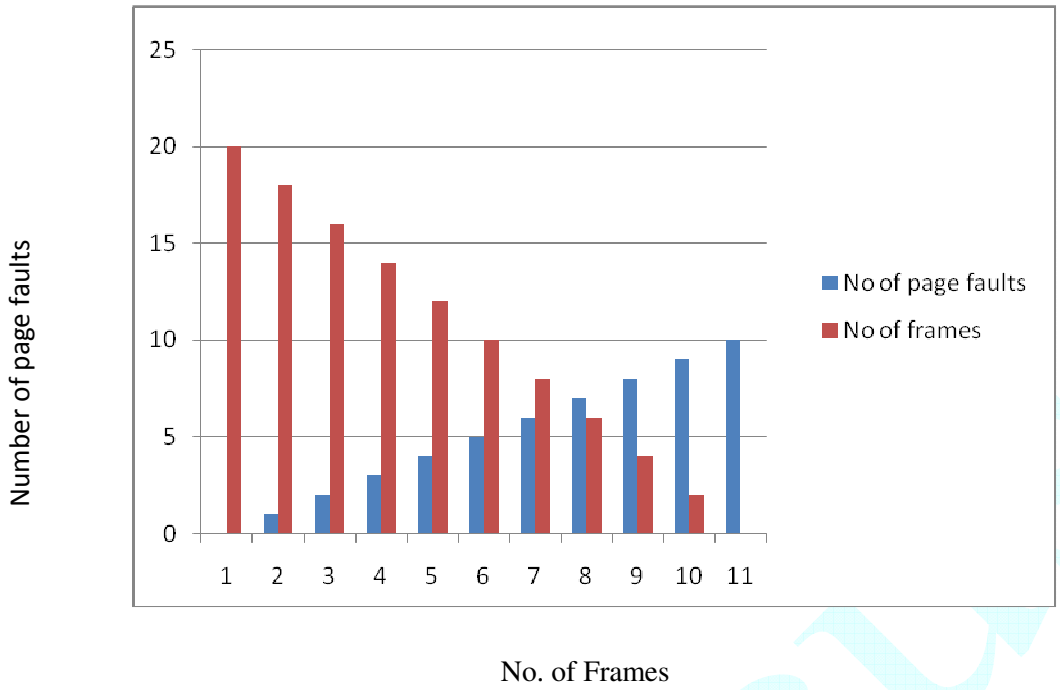

Figure 5: Page Fault Frequency

5. CONCLUSION

When a program needs a page that is not in the main memory, the operating system copies the required page into memory and copies another page back to the disk and this is referred to as paging. It is the page – in rate that is of primary concern because a page – in incurs a time cost for the physical I/O and a more significant increase in processor usage. Depending on the configuration and algorithms involved, the throughput and Latency of a system may degrade by multiple orders of magnitude. Hence, the battle cry resonates with a growing and wide spread concern for effective system performance. In this paper, analytical model, composed of a set of formulas and computational algorithms provides the values of desired performance measures as a function of the set of values of the performance parameters. Lyaponuv stability criteria provides a sufficient condition for the system validity. Interestingly, the result shows that the savings generated from the page- based, offsets the additional time needed to process the increase number of page- faults even in a memory constrained environment.

Terminologies
The term **workload** designates all the processing requests submitted to a system by the user community during any given period of time.
Throughput is the number of jobs that can be completed per unit time.
Residence time is the time spent by the job once it is loaded into memory.

The measure of load is given by the average number of batch jobs that are concurrently in execution and this number is called the average degree of **multiprogramming** and is denoted by $\bar{N}$.

Notations
K= number of devices
R = number of classes of customers,
$N_r = class.r.population$
$S_{i,r}$ = average service time of class r – customers at device i
$D_{i,r}$ = average device demand of class r- customers at device $i; D_{i,r} = V_{i,r} S_{i,r}$
$R_{i,r}$ = average response time per visit of class r customers to device i,
$R'_{i,r}$ = average residence time of class r customers at device i,
$\bar{n}_{i,r}$ = average number of class r customers at device i,
$X_{i,r}$ = class r throughput at device i,

REFERENCES

[AEP11] **Taha Ahmadzadeh, Hossein Etemadi, Ahmed Pifeh -** *Exploration of Factors Influencing on Choice of the Activity – Based Costing System in Iranain Organization*, International Journal of

Business Administration, Vol. 2, No. 1, 2011.

[BD06] **Haran Boral, David J. Dewitt -** *A Methodology for DtataBase System Performance Evaluation*, Computer Science Department, University of Wisconin Madison, 2006.

[B+06] **Qi Bi, Pi-Chun Chen, Yang Yang, Qinqing Zhang -** *An Analysis of VOIP Services Using IXEV – DO revision of a system*, IEEE Journal on Selected Areas in Communications, Vol. 24, No. 1, pages 36 – 45, 2006.

[KGP89] **R. Katz, G. Gibson, D. Patterson -** *Disk System Architecture for High Performance Computing*, Proceedings of the IEEE, Vol. 77, No. 12, Dec. 1989.

[MA82] **D. A. Menasce, V. A. Almeida -** *Operational Analysis of Multiclass Systems with Variable Multi-programming Level and Memory Queuing*. Computer Performance, Vol. 3, No. 3, Sept, 1982.

[MAD94] **Daniel Menasce, Virgilo A.T. Almelda, Larry W. Dowdy -** *Capacity Planning and Performance Modeling*, Prentice Hall, Inc. A Simon and Scheater Company, Eaglewood Cliff, new Jersey 07632, pages 64 – 210, 1994.

[MLT12] **Heng Ngee Mok, Yeow Leong Lee, Wee Kiat Tan -** *Setting up a Low-Cost Lab Management System for a Multipurpose Computing Lab Using Virtualization Technology*, Australasian Journal of Educational Technology, 28(2), pp. 266 – 278, (2012).

[WW00] **Allen Wynn, Jie Wu -** *The Effect of Compression on Performance in a Demand Paging Operating Systems and Software*, The Journal of Systems and Software, 50 (2000), pp. 151 – 170.

[ZNQ93] **Xiaodang Zhang, Nagas Nalluri, Xiaohan Qin -** *A Tool for Monitoring and Visualizing Min-Based Multi-processor Performance*, Journal of Parallel and Distributed Computing, 18, pp. 231 – 241, 1993.

Tibiscus