Software optimization technique for the reduction page fault to increase the processor performance

Jisha P.Abraham[#1], Sheena Mathew[*2]

[#]Department of Computer Science, Mar Athanasius College of Engineering, Kothamangalam,Kerala,India

[1] Jishapa@mace.ac.in

[*] Division of Computer Science
School of Engineering,Cochin University of Science and Technology, Kochi,Kerala,India

[2] Sheena Mathew @cusat.ac.in

***Abstract*—This work is aimed on how to increase the processor performance without adding any extra hardware. As technology grows new hardware are introduced and that replaces the old one. Which actually leads to the E-waste generation, one of big environmental problem. In the entire work no extra hardware is introduced inorder to increase the system performance. Processor performance can measured in terms of how much time the processor is spending to perform useful work. This is accomplished with the various stages of operations. The disk scheduling is done with MBFQV2 instead of CFQ. Page replacement method which is used for loading the new requested page to the main memory from virtual or loading to cache from main memory is replaced with the LRU_LFU scheduling. Along with the above two steps for handling the branch instruction a new method called CaMMEBH introduced. The increase in the processor performance is measured in terms of page fault. Even if there is some change in the various time parameters which are not notable. The main reason behind this is execution will take place only in sequentially. None of the compiler will generate the parallel execution code. By keeping the mentioned thing in mind here some modification are done on the operating system design itself..**

Keyword-MBFQV2 , LRU_LFU, CaMMEBH, Page Fault, Performance

I. INTRODUCTION

Disk scheduler is the one which is responsible for the transferring of data from secondary to the primary memory. It work based one principles of round robin scheduling. Where time slice will be given to all the request which will come to the scheduler list. The accessing will be done based on the queue data structure. Pre-emption process is activated inside the system in order to maintain the CFQ[1] concept, each request will get a fair allotment. The problem present in CFQ is that the arrival of a synchronous request may be arbitrarily delayed by a scheduler by just delaying the dispatching of the preceding request. Only because of the delayed invocation the request may get higher time stamp . The delay may be arises due to some other reason and if it activated earlier they may get better time stamp. Only because of higher time stamp the request have to wait in the request queue for long time period. Instead of setting the time slot a new concept of bandwidth is introduced in the BFQ[1,2]. But here also the allotment of slot it kept as static it means once the budget is fixed it will remain the same for the entire operation. Also the B max is fixed as the maximum budget value. Due to this the request with less budget have to wait for long time period. These two things are taken in consideration and modification is done on the BFQ method and formed MBFQV2[3]

Once the data made available in the primary memory, the next level of data movement is from primary to cache memory region. Whenever the data have to make a move to the next level before moving the data the availability of the space/room should checked. The space is known as page frame in the case of virtual memory to primary and from primary to cache it is cache line. Once the space is available then only the movement will take place. If there is no space first the space should be generated by replacing on of the existing page called victim page. This is done by the page replacement scheduler. The working of the scheduler can be done based on various algorithms. LRU is most commonly used replacement algorithm in almost all operating system. All the replacement algorithm will either take the arrival time of the page to the memory or how many number of time the page is referred by the system (the pages which are residing inside the memory). New replacement algorithm is suggested by modifying current LRU algorithm , the time stamp is mixed with the concept of frequency parameter in terms of distance. By making this modification page fault both major and minor will

get reduced and the performance of the system get increased. Whenever a page fault occur the processor have to spent a lot of time inorder to handle it. If it is major fault the processor cycle required is more. That is why major fault is more expensive than minor fault. The current LRU is replaced with LRU-LFU [6,7]in the kernel

One of the reason behind the page fault is the problem caused by the branch handler. Even if a number of branch prediction methods are available , none of them are completely error free. Branch handler will give the preference to the true condition only in the existing system. Modifying the current one with a new concept CaMMEBH[8], equal weightage will be given to both true and false condition. For this existing data structure which is used by the handler is modified by adding a new field to it which is able to handle the false condition. By this the handler will be ensure that the pages which are required by both true and false condition should present in the cache memory and primary memory.

II. PROPOSED SYSTEM

In the current system is make use of the CFQ for the disk scheduling , LRU for the page replacement method and the adaptive prefetching is make for the prefetching method. In the proposed system instead of CFQ we make use of the MBFQV2 is used for the disk scheduling. In this method the application work is in the service domain and not in the time domain, they are scheduled by the BFQ scheduler as a function of their budgets. Regardless of the time interval, workload and disk physical parameter, BFQ guarantees to each application a minimum possible delay, achievable by serving applications budget-by-budget. BFQ the budget (Bmax) will be always set as the maximum value of the budget in the request queue and kept it as static value. Which will lead to the starvation of the smaller processes. In order to overcome the drawback the allocation of the budget is done in a dynamic way. To make Bmax allocation in a dynamic form new Bmax is calculated by considering all the applications which are waiting in the request queue. Average budget of all the applications present in the request queue each time is calculated and is set as the Bmax value. If the average value is greater than threshold value then the threshold value with maximum budget is set. Whenever a process is completed the budget calculation will be reactivated by removing the serviced one from the request queue. Along with these steps C-LOOK scheduling concepts are also considered for the scheduling process, in order to reduce the rotational latency. The algorithm used for the implementation of BFQ is given in algorithm 1. and the calculation of the budget value is described under algorithm 2.

ALGORITHM 1 :-

Step1: input: application index, request issued by the application
- step 1.1 insert the request in the request queue
  - a d d r e q u e s t (i n i t , r e q u e s t R)
  - a p p l = a p p l i c a t i o n s [i] ; / / reference to the i-th application
- step 1.2 activate the timer for waiting time calculation

Step 2: Generate the tree structure for the storage of the request

Step 3: Generate the weighted tree corresponding to the request tree
- Step 3.1 Activate the sector calculation function(budget calculation)
- Step 3.2 assign the corresponding budget to various requests

Step 4: select the request for service
- Step 4.1 Calculate the sector distance from the current position to the request.
- Step 4.2 Fix the budget value for the request
- Step 4.3 Dispatch the request
  - r e q u e s t d i s p a t c h ( )

Step 5: Activate data transfer function

step 6: Activate the timer function

Step 7: check the status of the dispatch request
- Step 7.1 If the request is serviced completely
  - step7.1.1 the request should be removed from the application queue
  - step 7.1.2 the request should be removed from the tree
  - step 7.1.3 the request should be removed from the weighted tree
- Step7.2 if it is not serviced completely
  - step7.2.1 calculate the remaining budget value
  - step7.2.1 store the information about how much is serviced

Step8: Extract the next active application from queue
- Step 8.1 go to step 4.3

ALGORITHM 2 :-

```
defaultmaxbudget =16*1024
update_maximum_budget()
{
Budget =0;
```

```
For(i=0…maximum_queue_size)
{
Budget =Budget+Application.Budget;
Application++;
}
B_max =Budget/queuesize;          //queuesize  is the number application
                                  //currently in queue
If   (B-max>defaultmaxbudget)
{
 B_max =defaultmaxbudget;
}
```

The page replacement is done based on the LRU-LFU algorithm. Performance of the page replacement algorithm is measured in terms of major and minor fault[4,5]. Minor page fault is the one when a process attempts to access a portion of memory before it has been initialized. That is, a potential source of memory latency is called a minor page fault. When this occurs, the system will need to perform some operations to fill the memory maps or other management structures. The severity of a minor page fault can depend on system load and other factors, but they are usually short and have a negligible impact. The fault which is going to occur in the cache memory will come under this fault level.

Major fault occurs when the system has to synchronize memory buffers with the disk, swap memory pages belonging to other processes, or undertake any other Input/output activity to free memory. When the processor references a virtual memory address that didn't have a physical page allocated to it, the reference to an empty page causes the processor to execute a fault, and instructs the kernel code to allocate a page and return, all of which increases latency dramatically. That is the major fault occurring in the primary memory level. Due to this reason itself major fault is more expensive than minor fault. In this algorithm instead of using one parameter this method makes use of both the time (when) and frequency (how many). The LFU-LRU, need to consider the existing parameters available in LRU along with new parameter for frequency (freq).

For the handling of the branch instruction CaMMEBH is make used. CaMMEBH is a branch handling method which will make use of both the concept of adaptive prefetching (software prefetching) along with branch target prefetching(hard ware prefetching) methods. The branch handling are mainly focused on the conditional statements and the looping constructs in which the execution flow changes from the sequential manner. When the execution flow changes from a sequential manner, the chance of increasing page fault in cache memory is more and it reduces the throughput of the system. *CaMMEBH* reduces the page fault in memory through prefetching. Its objective is to load the next instruction that follows a branch handling jump instruction onto memory while executing the jump.

III. IMPLEMENTATION OF VARIOUS PHASES

The proposed system can be implemented through step by step procedure. It includes the installation and compilation of Linux-3.13 with modified program code. Kernel building involves creation, loading and installing of modules and these steps are explained with the use of commands make, make modules, make modules install. Make command is the one which does building or compiling of program. Loading of module is achieved by the next command make modules. The installation performed by the command make modules install.

IV. PERFORMANCE ANALYSIS

Table 1, Table 2 and Table 3 will give the details of change in major and minor fault and observation regarding the various time parameters. The work is mainly concentrated in the improvement of system performance. For that keep the processor free from the fault handling by reducing the major and minor fault rate

Table 1 the details about the major fault in various condition. Here the various scenarios are taken for the observation. In the first case only the CaMMEBH is considered. Results shows that the number of major fault is not increasing in any case but in certain cases the fault remains the same. The reason behind this is there is no major fault is generated inside the program due to the branching statements. CaMMEBH is not able to reduce any major fault generation other than branching. To handle the fault generation due to other condition, other methods should be included along with CaMMEBH. CaMMEBH is combined with LRU-LFU, the results are shown in the Table 1. Number of major fault is reducing in all the cases. The reason behind this may be due to replacement algorithm LRU in which how the selection of the vitim page is done. If the vitim page selected by the LRU is the one which is referred by system in the next reference, it lead to major fault. This selection method is replaced in the case of LRU_LFU.

TABLE I DETAILS OF MAJOR FAULT

FILE SIZE	CURRENT	CaMMEBH	CaMMEBH+LRU -LFU	CaMMEBH +MBFQV2	CaMMEBH+LRU -LFU+MBFQV2
300KB	7	5	5	4	3
350KB	6	4	3	3	2
912KB	3	1	1	1	1
1.43MB	6	5	4	4	4
2MB	9	8	6	5	3
15MB	5	4	4	4	3
20MB	23	20	20	29	16
25MB	3	2	2	2	1
65MB	24	21	21	20	18
70MB	115	108	100	98	95
75MB	54	49	50	50	47
78MB	56	52	51	50	48

CaMMEBH is merged with MBFQV2, the results are shown in Table 1. This case also the number of major fault is reducing in all the cases. The reason behind this may be how fast the pages are made available in the respective memory levels. If the corresponding page is not available in the memory it should be taken from the lower memory level. This will depend on how fast the data can move from one level to the other one. Even if the chance of occurring the fault is detected and the page can't be bring to the main memory it leads to the major fault. This condition is handling in this scenario.

Finally CaMMEBH is merged with both LRU_LFU and MBFQV2, the results are shown in Table1. Here also in all the cases the major fault is get reduced. Fault due to branching, improper page replacement method and delay due to the waiting in the request queue for transferring the page from secondary to primary are considered in this case. Results shows that there is a drastic change in the major fault number in certain cases. But the major fault can't be completely removed from the system. The graphical representation of Table1 is given in Figure 1

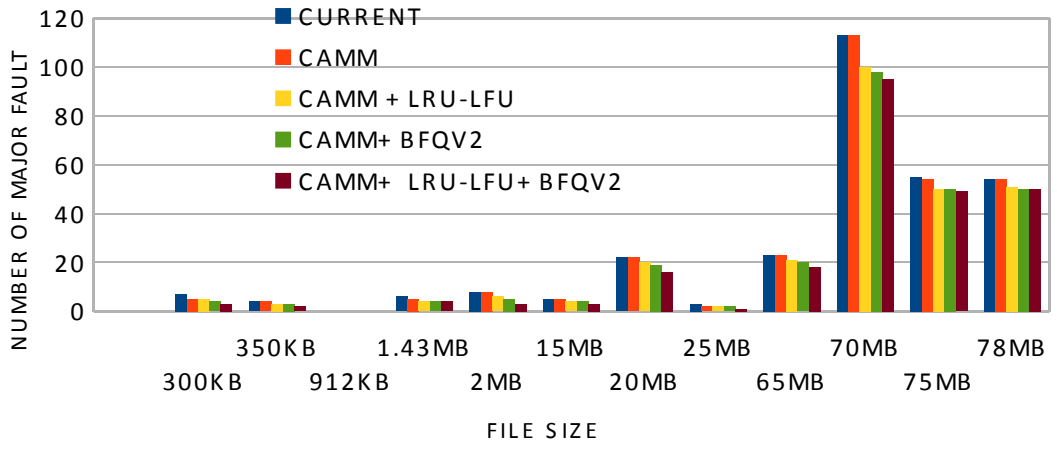

Figure 1: Comparison of major fault in various scenario.

Table2 will give the details about the minor fault. In this cases also the various condition are considered as in the case of major fault.

In the first case only the CaMMEBH is considered. Results shows that the number of minor fault is not increasing in any case but in certain cases the fault remains the same. The reason behind this is there is no minor fault is generated inside the program due to the branching statements. CaMMEBH is not able to reduce any minor fault generation other than branching. To handle the fault generation due to other condition, other methods should be included along with CaMMEBH. CaMMEBH is combined with LRU-LFU , the results are shown in the Table 2. Number of minor fault is reducing in all the cases. The reason behind this may be due to replacement algorithm LRU in which how the selection of the vitim page is done. If the vitim page selected by the LRU is the one which is referred by system in the next reference, it lead to minor fault. This selection method is replaced in the case of LRU_LFU.

TABLE II DETAILS OF MINOR FAULT

FILE SIZE	CURRENT	CaMMEBH	CaMMEBH+LRU-LFU	CaMMEBH +MBFQV2	CaMMEBH+LRU-LFU+MBFQV2
0KB	6998	6960	6583	6815	6557
350KB	2161	204	1624	1635	1478
912KB	5139	5116	5110	5205	5104
1.43MB	5601	5510	5837	5680	5456
2MB	240	238	239	237	237
15MB	240	238	239	239	237
20MB	239	238	238	238	236
25MB	239	237	237	237	238
65MB	27481	26457	25783	25233	24346
70MB	36586	33724	33653	33862	33242
75MB	35873	34456	33686	34327	33456
78MB	36487	35343	34586	33726	34543

CaMMEBH is merged with MBFQV2, the results are shown in Table 2. This case also the number of minor fault is reducing in all the cases. The reason behind this may be how fast the pages are made available in the respective memory levels. If the corresponding page is not available in the memory it should be taken from the lower memory level. This will depend on how fast the data can move from one level to the other one. Even if the chance of occurring the fault is detected and the page can't be bring to the cache memory it leads to the minor fault. This condition is handling in this scenario.

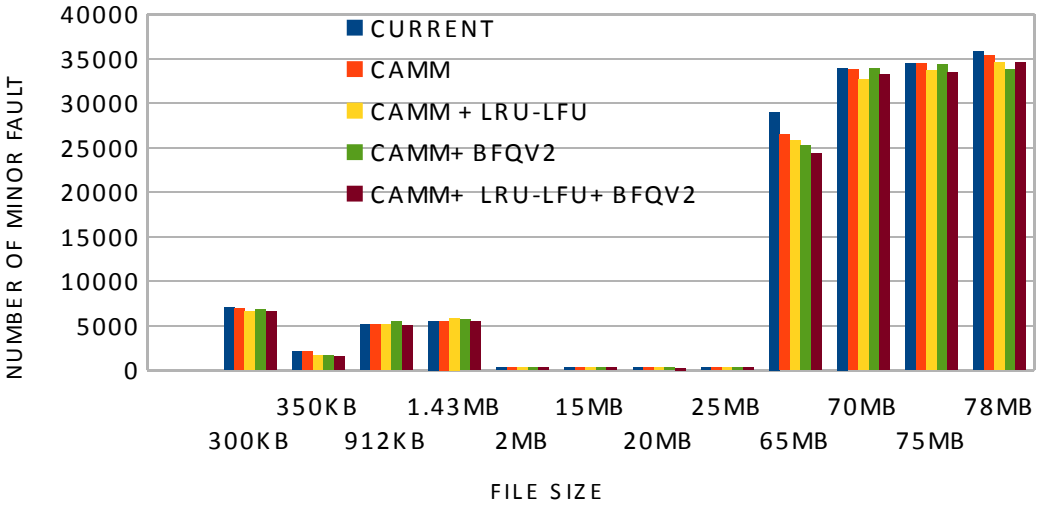

Figure 2: Comparison of minor fault in various scenario

Fnally CaMMEBH is merged with both LRU_LFU and MBFQV2, the results are shown in Table1. Here also in all the cases the minor fault is get reduced. Fault due to branching, improper page replacement method and delay due to the waiting in the request queue for transferring the page from secondary to primary are considered in this case. Results shows that there is a drastic change in the minor fault number in certain cases. But the minor fault can't be completely removed from the system. The graphical representation of Table 2 is given in Figure 2.

TABLEIII DETAILS OF TIME PARAMETERS

FILE SIZE	CURRENT		CaMMEBH		CaMMEBH+LRU-LFU		CaMMEBH +MBFQV2		CaMMEBH+LRU-LFU+MBFQV2	
	User time (sec)	System time (sec)	User time (sec)	System time (sec)	User time (sec)	System time (sec)	User time (sec)	System time (sec)	User time (sec)	System time (sec)
300KB	00.01	0.041	00.01	0.042	00.01	0.042	00.01	0.041	00.01	0.04
350KB	00.00	0.042	00.00	0.043	00.00	0.042	00.00	0.04	00.00	0.042
912KB	00.00	0.042	00.00	0.042	00.00	0.041	00.00	0.042	00.00	0.041
1.43MB	00.01	0.041	00.01	0.041	00.01	0.043	00.01	0.041	00.01	0.041
2MB	00.00	0.043	00.00	0.041	00.00	0.043	00.00	0.041	00.00	0.042
15MB	00.01	0.042	00.01	0.042	00.01	0.041	00.01	0.042	00.01	0.041
20MB	00.00	0.042	00.00	0.042	00.00	0.042	00.00	0.042	00.00	0.042
25MB	00.00	0.041	00.00	0.041	00.00	0.041	00.00	0.043	00.00	0.04
65MB	00.00	0.043	00.00	0.041	00.00	0.042	00.00	0.043	00.00	0.043
70MB	00.01	0.041	00.01	0.043	00.01	0.041	00.01	0.042	00.01	0.041
75MB	00.00	0.043	00.00	0.043	00.00	0.042	00.00	0.041	00.00	0.042
78MB	00.01	0.041	00.00	0.041	00.00	0.042	00.01	0.042	00.01	0.041

Table 3 the details about time parameters in various condition. Here the various scenarios are taken for the observation. The results shows that even if the correction is made on the data transfer method, page replacement method or the branch handler it does not affect the user time or system time parameters. By the above methods the fault rate can be reduced, but it will affect the execution parameter. Even if all the operating systems are designed with multi-threading capability the compilers are not able to utilise this feature. The execution will be done in sequential order only. That is reason behind there is no change in the user time or in

the system time parameters. If the execution speed has to increase the facility of multithread should be utilized in proper way .

V. CONCLUSION

The results shows that by incorporating these various methods in the current Linux version number of major and minor fault can be reduced . If the fault rate is get reduced the performance of the system will get increased by freeing the processor for fault handing condition. Analysis of the time parameter shows that even if the number of major and minor fault is get decreased the execution time is get reduced. This is because of the compilers which we are make used. Modern processors are able to handle multiple threads in parallel. But the compilers are not able to initiate multiple threads. This will affect the execution time parameters. Experiment results shows that the waiting time is get reduced by the above methods.

REFERENCES

[1] Paolo Valente and Fabio Checconi"High Throughput Disk Scheduling With Fair Bandwidth Distribution" IEEEtransactions on computers. Vol 59,no.9,september 2010
[2] s.layer and p. Druschel "anticipatory scheduling: A Disk Scheduling Framework to overcome Deceptive Idleness in synchronous/O", Proc.18th ACM Symp. Operating systems Principles, Oct 2001
[3] Jisha P Abraham, Sheena Mathew "High Throughput disk scheduling with equivalent bandwidth sharing" Indian Journal of Science And Technology(Journal extension -IEEE International conference on Innovations in Information, Embedded and Communication Systems)
[4] Sedigheh Khajoueinejad, Mojtaba Sabeghi, AzamSadeghzadeh. A Fuzzy Cache Replacement Policy and its Experimental Performance Assessment, 2006 IEEE..
[5] Elizabeth J O'Neill ,Patrick E O'Neill,Gerhard Weikum The LRU-K page replacement Algorithm For Database Disk Buffering SIGMOD Washington, DC,USA 1993 ACM.
[6] Jisha P Abraham, Sheena Mathew "A novel approach to improve the processor performance with page replacement method". International Conference on information and communication Technology (ICICT-2014)
[7] Jisha P Abraham, Sheena Mathew "A novel approach to improve the system performance by proper scheduling in memory management" 2nd international conference on Emerging trends in electrical, communication and information technologies 2015.
[8] Jisha P Abraham, Sheena Mathew "Efficient CaMMEBH for page fault reduction" International Journal of Applied Engineering Research.