Energy Efficient Virtual Distributed Scheduling in a Multi-processor Environment

Prof. P. M. Bihade
Department of Information Technology
S.R.P.C.E.
Nagpur, India
pmbihade29@gmail1.com

Pro. Hemlata R. Kosare
Department of Information Technology
S.R.P.C.E.
Nagpur, India
patilhema5@gmail

Abstract— The increase in commercial and industrial applications of embedded and real-time systems in the last few decades required the need of multiprocessor systems to cater to these complex applications and computations. However these multiprocessor systems often consume substantial amount of energy, hence minimizing energy consumption is one of the most challenging topics for the design of embedded and real-time systems using chip-multiprocessors. One way to minimize the energy consumption is to employ scheduler to distribute the load among the multi-processor in an efficient manner. In this paper we propose an optimized virtual distributed scheduler based energy efficient multi-processor system.

Keywords*— Multi-processor Systems, Load Balancing, Quality of Service (QoS), Scheduler.*

*****

I. INTRODUCTION

With the increase in complex applications, softwares and services, which often requires large and complex computations came the need multiple processor system. The main aim behind incorporating a multiprocessor system is the desire to increase the speed of execution of workload. Parts of the workload, referred to as tasks, can be distributed among various processors and thus can be executed more quickly than on a single processor. This optimized system can be achieved only through the help of a scheduler, where a scheduler assigns the tasks to different processors available in the system. The scheduling problem for multiprocessor systems can be generally stated as "How can we execute a set of tasks T on a set of processors P subject to some set of optimizing criteria C?" [1]. Thus the main goal of the scheduler is to minimize the expected runtime of a task set. The energy-efficient scheduling problem for uni-processor and multiprocessor in real-time systems has been an important field for researchers in the past few years [2].

There are two major types of scheduling.

A. *Local scheduling* – It is performed by the operating system for the uniprocessor.
B. *Global scheduling* – It decides where to execute a process in a multiprocessor system. Global scheduling can be achieved through a single master processor, or by distributed among the processors.

Global scheduling can be further classified as static or dynamic scheduling. In static scheduling processes are assigned to processors before the execution starts. In simple terms static scheduling algorithms require information about individual processes only. Dynamic scheduling on the other hand can reassign the processes to the processors during the execution and may take processor's current load in consideration.

In this paper a review the different scheduling algorithms has been presented. Finally our proposed optimized virtual distributed scheduler based energy efficient multi-processor system is presented. The rest of the paper is organized as follows: Section 2 presents literature review and discusses the advantages of dynamic load balancing over static algorithms. In Section 3 we discuss some key load balancing algorithms. Section presents our proposed system. Section 4 gives the proposed distributed scheduler based energy efficient multi-processor system and finally, Section 5 concludes and provides some future directions.

II. LITERATURE SURVEY

Load balancing is a special case of load sharing, in which the goal of the global scheduling algorithm is to keep the load even across all processors. Baumgartner and Wah [3] in his work has presented a systematic characterization of load balancing strategies considering the different methodology such as static, dynamic, adaptable etc, and considering different properties such as cooperation, location of control, initiation, etc.

In static load balancing (SLB) algorithms the performance of the processors is determined at the beginning of execution and no dynamic information is used. Depending on their performance such as average execution time, arrival time, resources required, etc, the workload is distributed right at the start by the master processor. The slave processors then simply do the work assigned to them and submit their result to the master. Thus SLB methods reduce the overall execution time of a concurrent program and minimize communication delays. However the major drawback of SLB methods is that in certain cases the complete required information may not be available the allocation time and thus an assigned process to a processor cannot be changed during process execution to make changes in the system load i.e., SLB algorithms are non-preemptive. Round Robin Algorithm, Randomized Algorithm, Rate-monotonic, Central Manager Algorithm, etc are some of the common examples of SLB algorithms.

In Round Robin algorithm, the processes are evenly divided among all processors, whereas in Randomized algorithm a processor for a process is selected randomly. In Central Manager Algorithm, a central or master processor selects the node for each new process. In this algorithm, high degree of inter-process communication may lead to bottleneck situation [4].

In Dynamic load balancing (DLB) algorithms the work load is distributed among the processors at runtime. Unlike static algorithms, dynamic algorithms buffer the processes in the queue on the main node and allocated dynamically upon requests from remote nodes. Earliest-deadline-first (EDF), least-laxity-first (LLF), FPZL and DPZL are the examples of DLB algorithms. DLB algorithms can be further classified into following categories [5]:

1. *Sender Initiated (SI)*

In SI algorithms, a heavily loaded node initiates the load balancing process by requesting their present load information from other nodes and then assigning the current task to the lightly loaded nodes. Thus SI algorithms facilitate process migration from a heavily loaded node to a lightly loaded node. Three basic decisions that are needed to be considered before making a transfer of a process are [5]:

- Transfer policy: When does a node become the sender?
- Selection policy: How does a sender choose a process for transfer?
- Location policy: What node should be the target receiver?

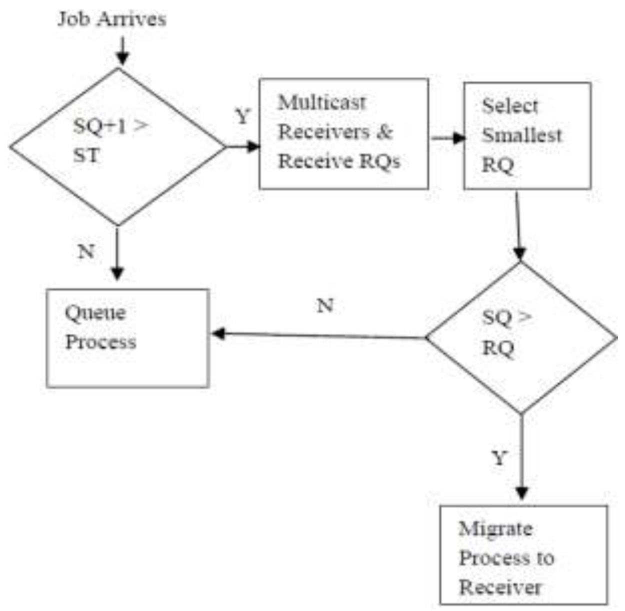

Figure 1: Flowchart of the Sender-Initiated Algorithm [6].

Figure 1 above depicts the flowchart of the sender-initiated algorithm, where ST, SQ and RQ depict the sender threshold, queue length of the sender and receiver respectively. The sender sends a multicast message asking for the queue sizes of all other nodes. Upon receiving this, the sender can select the node with the smallest queue length (RQ) as the target receiver, provided that SQ>RQ. This exchange of multicast messages and migration of processes between senders and receivers brings upon additional communication overhead, which increases the actual load of the system.

2. *Receiver Initiated (RI)*

The receiver initiated algorithms use a similar transfer policy as sender-initiated algorithm, which activates the pull operation when its queue length falls below a certain threshold (RT), upon the departure of a job. Figure 2 depicts the flowchart of the receiver-initiated algorithm.

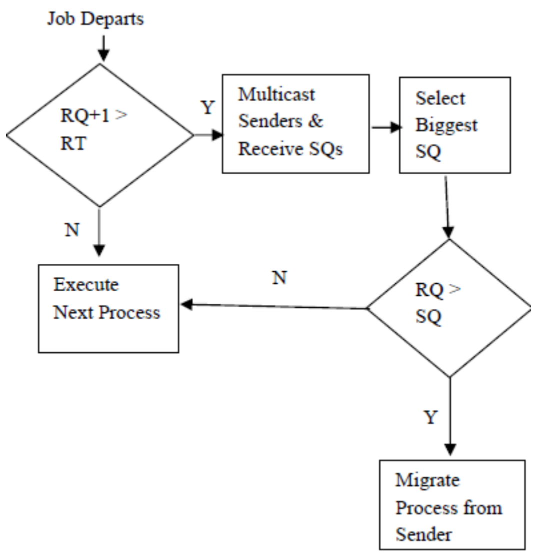

Figure 2: Flowchart of the Receiver-Initiated Algorithm [6].

3. *Symmetrically Initiated (SI)*

As discussed above that both sender-initiated and receiver-initiated algorithms work extremely well at different system load conditions, a combination of both the techniques can be advantageous. A node can activate the sender-initiated algorithms when its queue size exceeds one threshold ST, and can activate the receiver-initiated algorithm when its queue size falls below another threshold RT. Thus each node can dynamically play the role of either a sender or a receiver.

4. *Periodically Exchanged (PE).*

In the PE algorithms, nodes periodically exchange their load information and use the load information of last period as the reference to make Process transfer decisions for the current period. When a new process comes, a node checks whether its current load level is too high. If so, it randomly selects a number of nodes, up to a preset limit, whose last period load level are low enough, and polls them one by one. The new process is transferred to the first node whose current load level is low enough. If no such node existed, the job is processed locally [6].

Carpentar et al. [7] suggested the categorization shown as in Table 1. These load balancing algorithms can be further classified on the basis of degree of migration allowed as follows: (i) no migration (i.e., task partitioning), (ii) migration allowed, but only at job boundaries and (iii) unrestricted migration. The entries in Table 1 give known schedulable utilization bounds for each category, assuming that jobs can be preempted and resumed later. If U is a schedulable utilization for an M-processor scheduling algorithm *A*, then *A* can correctly schedule any set of periodic (or sporadic) tasks with total utilization at most U on M processors.

TABLE I: Known lower and upper bounds on schedulable utilization for the different classes of pre-emptive scheduling algorithms [7].

3: full migration	$\frac{M^2}{3M-2} \le U$	$U \ge M - \alpha(M-1)$, if $\alpha \le \frac{1}{2}$ $\frac{M^2}{2M-1} \le U \le \frac{M+1}{2}$, otherwise	$U = M$
2: restricted migration	$U \le$ M	$U \ge M - \alpha(M-1)$, if α $M - \alpha(M-1) \le U \le$ M ,otherwi	$U \ge M - \alpha(M-1)$, if $\alpha \le \frac{1}{2}$ $M - \alpha(M-1) \le U \le \frac{M+1}{2}$,otherwise
1: partitioned	$(\sqrt{2}-1)M \le$	$U = \frac{\beta M+1}{\beta+1}$, where $\beta =$	$U = \frac{M+1}{2}$
	1: static	2: job-level dynamic	3: fully dynamic

III. Scheduling Algorithms

1. *First Come First Serve (FCFS):*

FCFS is the simplest way for a scheduler to schedule the packets; it just assigns the tasks according to the order of their arrival time. Thus, the QoS guarantee provided by FCFS is in general weak and highly depends on the traffic characteristic [8]. Its a non-preemptive algorithm that treats ready queue as FIFO. However, since FCFS has the advantage of simple to implement and has low overheads, it is still adopted in systems providing best effort services.

2. *Round Robin (RR):*

Round Robin (RR) is one of the oldest, simplest and most widely used scheduling algorithms that offers better bandwidth utilization and compensates the drawbacks of FCFS [9]. Its basically FCFS with Preemption. The scheduler polls each flow queue in a cyclic order and serves a task from any-empty buffer encountered; therefore, the RR scheme is also called flow-based RR scheme. Turnaround time is typically larger for RR than SRTF but has better response time. However the performance depends on quantum q: Small q values causes overhead due to context switches (& scheduling) and in case of large q values the algorithm behaves like FCFS.

3. *Shortest Remaining Time First (SRTF):*

SRTF is a scheduling method that is a preemptive version of shortest job next (SJN) scheduling. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time. Shortest remaining time is advantageous because short processes are handled very quickly. The system also requires very little overhead since it only makes a decision when a process completes or a new process is added, and when a new process is added the algorithm only needs to compare the currently executing process with the new process, ignoring all other processes currently waiting to execute.

4. *Earliest Deadline First (EDF):*

This is the most widely used scheduling algorithm to minimize queuing delay in real time video traffic. The most important and analyzed dynamic priority algorithm is Earliest Deadline First (EDF). The priority of a job (instance) is inversely proportional to its absolute deadline, in other words the highest priority job is the one with the earliest deadline. If two tasks have the same absolute deadlines, chose one of the two at random (ties can be broken arbitrarily). The priority is dynamic since it changes for different jobs of the same task. One of the problems with EDF is that the low priority queues get starved. Here in EDF the better throughput is achieved without taking fairness constraints. Under EDF discipline, each flow is assigned a tolerant delay bound d_i; a packet j of flow i arriving at time a_{ij} is naturally assigned a deadline $a_{ij}+d_i$ [10]. Each eligible packet is sent according to the increasing order of their deadlines.

5. *Deficit Round Robin (DRR):*

Deficit-Round-Robin is a modified round robin algorithm that was originally developed for IP networks. In DRR, the deficit counter of each active connection is increased by quantum when the connection has its turn. If the size of the head-of-line packet of this connection is smaller than or equal to the deficit counter, we send the packet and decrease the deficit counter by the size of the packet [12, 13, 15]. We continue sending packets as long as the deficit counter allows. If the deficit counter is too small for the head-of-line packet, we move to the next connection. The deficit that is stored in the deficit counter of the connection is then saved for the next round. If we were able to serve all packets of the connection, the deficit counter is reset to zero. The accumulation of credits being unused will result in unfairness, thus the throughput is decreased.

6. *Multilevel queue scheduling:*

This is used for situations in which processes are easily divided into different groups. The system has several queues, each has a level which corresponds to a level of priority, each level has its own scheduler, there is one main scheduler which handles the scheduler for each queue. For example, a common division is made between foreground (interactive) processes

and background (batch) processes. This type of algorithm is very useful for shared memory problems. The only problem with this is that once a job is put on a queue, it must remain there until it completes. This could create some problems with CPU hogging and starvation.

7. Fixed Priority until Zero Laxity (FPZL)

FPZL is similar to global fixed priority preemptive scheduling, however, whenever a task reaches a state of zero laxity it is given the highest priority. FPZL is a minimally dynamic algorithm, in that the priority of a job can change at most once during its execution, bounding the number of pre-emptions. FPSL and FPCL are variants of FPZL that introduce no additional scheduling points beyond those present with fixed priority scheduling. FPSL, FPCL and FPZL are minimally dynamic algorithms, in that the priority of a job can change at most once during its execution, bounding the number of pre-emptions [11].

TABLE II: Comparison of scheduling algorithms

Scheduling algorithm	**CPU Overhead**	**Throughput**	**Turnaround time**	**Response time**
First In First Out	Low	Low	High	Low
Shortest Job First	Medium	High	Medium	Medium
Priority based scheduling	Medium	Low	High	High
Round-robin scheduling	High	Medium	Medium	High
Deficit Round Robin	High	High	Medium	High
Earliest Deadline First	High	High	Medium	High
Multilevel Queue scheduling	High	High	Medium	Medium
FPZL	High	High	High	High

IV. PROPOSED SCHEDULING SYSTEM

In this paper we propose an optimized virtual distributed scheduler based energy efficient multi-processor system as in figure 3 below. The proposed system consists of a scheduler, each new request or task is assigned to the scheduler first. The scheduler at each new request sends a multicast to all the processors asking their current load and assigns the task to the processor with minimum load. This simple load balancing can be achieved by using any of the dynamic algorithms such as EDF, SPZL, DPZL, etc. The next part that is incorporated in the proposed system is to involve energy optimization. This is achieved by dynamically considering the energy requirement of the task. Consider the case that the task is simple and each with minimum load. Now consider that the task at hand is complex and requires high level of computation, which can be achieved through certain processors in the system. Also let us assume that the high end processor that complete this task has the current load on the higher side. Then in such a case the scheduler assigns this task to the processor with the minimum load initially and periodically checks for the load conditions of the processor that can only solve this task. Once that this processor gets its load reduced, the scheduler takes this task from the initially assigned processor with its intermediate results and assigns the remaining task to the high end processor. Another way to achieve energy optimisation is that in case a processor reaches a state from where it is unable to process a particular task, then the same task with intermediate results obtained by this processor are passed onto the one with the minimum current load.

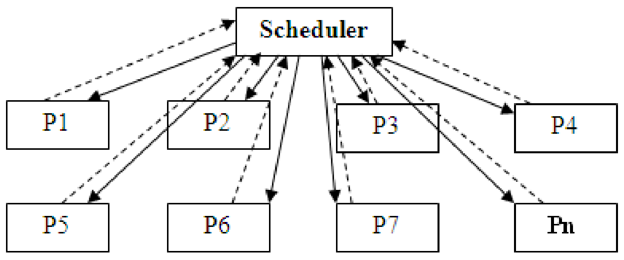

Figure 3: Proposed System Architecture.

V. CONCLUSIONS

In this paper, a survey of different scheduling algorithms is presented. Further we propose an optimized virtual distributed scheduler based energy efficient multi-processor system. The proposed system uses any of the dynamic algorithms available and incorporates the energy efficient mechanism to achieve energy optimization.

REFERENCES

[1] Steve J. Chapin and Jon B. Weissman, "Distributed and Multiprocessor Scheduling", Electrical Engineering and Computer Science Department, Syracuse University, 2002

[2] J. Chen and C. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms," in RTCSA'07, pp. 28–38.

[3] L. Sha and J. Goodenough. Real-time scheduling theory and Ada. IEEE Computer, 23(4):53{62, 1990.

[4] McEntire, P.L., O'Reilly, J.G., Larson, R.E.: Distributed Computing Concepts and Implementations, IEEE Press, New York (1984)

[5] Mahk, S.: Dynamic Load Balancing in a Network of Workstation. Research Report, (2000).

[6] Abirami M S, Niranjana G, "Dynamic Load Balancing in Distributed Systems", International Conference on Computing and Control Engineering (ICCCE 2012), ISBN 978-1-4675-2248-9, 2012.

[7] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. In Joseph Y. Leung, editor, Handbook on Scheduling Algorithms, Methods, and Models. Chapman Hall/CRC, 2004.

[8] Tsung-Yu Tsai, Yao-Liang Chung and Zsehong Tsai, "Introduction to Packet Scheduling Algorithms for Communication Networks"

[9] Tomá´l Balogh and Denisa Luknárová, Martin Medvecký, "Performance of Round Robin-based Queue Scheduling Algorithms", Third International Conference on Communication Theory, Reliability, and Quality of Service, 2010.

[10] H. Goto, Y. Hasegawa, and M. Tanaka, “Efficient Scheduling Focusing on the Duality of MPL Representatives,” Proc. IEEE Symp. Computational Intelligence in Scheduling (SCIS 07), IEEE Press, Dec. 2007, pp. 57-64, doi:10.1109/SCIS.2007.357670.

[11] Ashish Chandak, Bibhudatta Sahoo, Ashok Kumar Turuk, "Efficient Task Scheduling using Mobile Grid", IEEE-International Conference on Recent Trends in Information Technology, ICRTIT 2011

[12] Robert I. Davis and Shinpei Kato, "FPSL, FPCL and FPZL schedulability analysis", Journal Real-Time Systems archive Volume 48 Issue 6, November 2012 Pages 750-788

[13] Chirayu Nagaraju and Mahasweta Sarkar, "A Packet Scheduling To Enhance Quality of Service in IEEE 802.16", Proceedings of the World Congress on Engineering and Computer Science 2009 Vol I WCECS 2009, October 20-22, 2009, San Francisco, USA.

[14] J. Lopez, M. Garcia, J. Diaz, and D. Garcia, "Worst-case utilization bound for edf scheduling on real-time multiprocessorsystems", In Proceedings of the 12th Euromicro Conference on Real-time Systems, pages 25{33, June 2000.