The International Journal of Engineering
And Science (IJES)
||Volume|| 2 ||Issue|| 1 ||Pages|| 293-298 ||2013||
ISSN: 2319 – 1813 ISBN: 2319 – 1805

THE IJES

Amalgamate Scheduling Of Real-Time Tasks And Effective Utilization On Multiprocessors With Work-Stealing

[1]Sreenath.M, [2]Sukumar.P, [3]naganarasaiahgoud.K,
[1,]Asst. Professors, [2,]Dept. Of E.C.E.
[1,2,3,]Annamacharya Institute Of Technology And Sciences, Rajampet.

Abstract

The load balancing between resources in order to get effective utilization when many jobs are running with varying characteristics. The system speed is reduced when the work load is not properly balanced and idle processors are not involve for tasks execution, due to single processor may execute limit number of tasks. Thus the tasks steal from master processor to idle processors. Here work stealing is efficient approach to the distributed dynamic load balancing for the load among different processors. But Our proposal is priority based deque in place of the non-priority deque of work-stealing. Work stealing will increase the speedup of Parallel applications without affecting the schedulability of the other jobs scheduled by EDF-HSB.

Keywords: *deque,* EDF-HSB, *effective utilization, work stealing.*

Date of Submission: 29, December, 2012 Date of Publication: 30, January 2013

I. Introduction

The system speed is increased when the work load is properly balanced and idle processors are involve for tasks execution, due to single processor may execute limit number of tasks. Thus the tasks steal from master processor to idle processors. Here work stealing is efficient approach to the distributed dynamic load balancing for the load among different processors. Work stealing double-ended queues (deques) as a stack, push and pop tasks from the bottom, but idle processor deques stealing tasks from the top. Let us consider as an example for the task distribution.

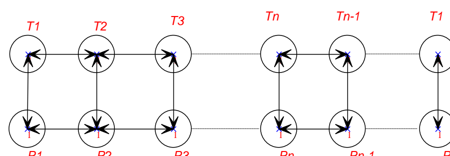


Fig 1: *Tasks read and write by the processors.*

T1, T2, T3 tasks are present for utilization and if any task T4 to Tn, insert task at the end of the queue. First step is T1 read and write by the processor P1, and T2 task is read and write T4 task when present in the queue. Simultaneously T3 task read and T5 task write by the processor. Accordingly the tasks are assigned to the different processors. If any task is not found, the process is going on without involvement of tasks T4 to Tn. Growing importance of parallel task models in real-time applications poses new challenges to real-time scheduling. Task-based parallelism enforced through compilers still lacks the ability to handle highly complex source code. The usefulness of existing real-time scheduling approaches is limited by their restrictive parallel task models. In contrast, the more general parallel task model addressed in our work allows jobs to generate an arbitrary number of parallel threads at different stages of their computations.

II. Work Stealing

One of the simplest, yet best-performing, dynamic load-balancing algorithms for shared-memory architectures. Uses a "breadth-first theft, depth-first work" scheduling policy with minimal overhead and good data locality. The challenge is to impose a priority-based (2) scheduling policy that positively increases the speedup of parallel applications without jeopardizing the schedulability of the system. The global distributed shared memory enables the tasks to be executed on any process in any processor during the computation. Work stealing is performed by storing Distributed task queue in the global distributed shared Memory (11). Work

stealing (3) is a scheme of dynamic load balancing (12). Each process follows a double-ended queue tasks. Tasks are executed from the deque, if any work is not found steal task from the processor's deque. The process that initiates the steal operation is called as "thief". The processor targeted by the steal is called "victim". The thief is reacting for initial load balance requests. The thief must select its victim first while performing the steal operation. Ones thief is selected the thief is fetch the information from the selected victim. Work is not assigned to the victim then the thief is selecting another victim randomly or priority based approach. This is global distributed shared memory (7) (GDSM) termination detection. The process must actively detect the all processes until work is available. This process is used to determine time period for computation. Work stealing scheduling algorithm collects data and distributed to different processes until tasks are available to seal. Hence it completes job in (t1/ (P+Tp)) expected time. Work stealing improves execution time and efficiency. System has large number of processors but many idle processors are stealing from busy processors. It tries to balance the loads on processors with additional stealing. The involvement of processors gradually develops in stealing. Finally conclude that the processors execution time.

Decreases and increase the efficiently (13) distribute for execute the tasks by using work stealing scheduling algorithms.

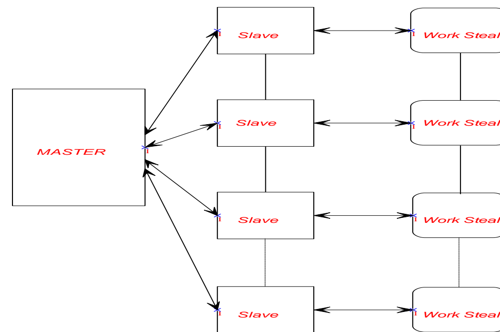


Fig 2: Master tasks distribution to slaves

Master having queue and it is used for control the all slaves in way of parallelism (4). The master main task is data distribution among different slave devices on the bases of functional, operational and performance specifications. Master takes decision regarding task distribution upon the work stealing information about task execution. Finally the master monitors the work load of the slaves and redistributes task whenever load imbalance is detected. The master is responsible for both scheduling and distribution of tasks, the allow slaves to complete data redundantly. This mechanism also makes the model toleralable to the failure of slaves.

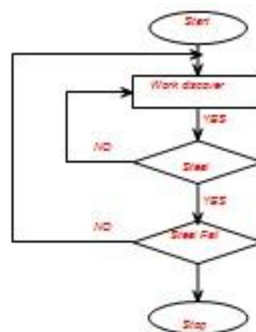


Fig 3: Flow chart for work stealing

2.1 Work Stealing Algorithm (DLBS)

Dynamic load balancing in distributed shared memory.

If Mpq the master processor queue
Spq be the slave processor task queue
Vsq be the victim slave task queue and
Tsq be the thief slave task queue
For slaves fromji=1 to n
Initialize all the tasks to Mq

Collect load status of slaves Spq
Tasks Distribute from Mpq to Spq
End for
Select thief Tj and victim Vj
Fetch work from victim's queue Vsq.
If work Wj found
Transfer tasks from Vsq to Tsq
Steal and execute work Wj.
Search for new task
Else
Steal Failed
Terminate steal operation
End If
End While

This scheduling approach for parallel real time execution of tasks and this type coexist with complex applications. This combines a multiprocessor residual capacity reclaiming scheme with a priority based work-stealing policy which, while ensuring isolation among tasks, allows a tasks to be executed in more than one processor at a given time.State based scheduling with tree schedules: analysis and evaluation dynamic scheduling of parallel computations by work stealing has gained good and desired responses. Work stealing has proven to be effective in reducing the complexity of parallel programming, especially for irregular and dynamic co0mputations. Thus the work stealing approach adopted in commercial and open source software and libraries. Work stealing has fixed number of workers means one per core. Double ended queue (deque) to store ready tasks. Deque acts as a stack push and pop the contents or tasks from the bottom of the stack, but treat the deque of another randomly chosen busy worker as a queue, stealing tasks only from the top whenever they have no local task to execute. The queue tasks are executed in a constant time independently work stealing support to execute the number of tasks basically, but main problem is tasks waiting for execution in a deque may be repressed by new tasks which are enqued at bottom of the deque.

A task at the tail of deque might never be executed if all processors are busy. There is no controlling for multi task execution. Task priorities and deque per core (SDPC) would require during stealing, the processor execute the tasks repeatedly in all deques until the highest priority task to be selected. This proposal gives valid solution to decrease the theft time. Work stealing load balancing policy used to allow parallel tasks to execute more than 1 processor at the same time instant.Each processor has a local deque to store tasks. Processor will try to select or select a task from the top of the other busy processor's deque while processor has no local tasks to execute. Here main important issue is which processor task will be selected. These choices bad to variations of the work stealing algorithm. Here more information not required about all processors due to the processor randomly select this is one of the benefits. Work stealing algorithm does not take deadlines in to account when task stealing from another processor. a task must block before another task can be scheduled on the same cpu or it will run to completion . Here more than task executed on multi processors. The local deques are performed in a last in first out order. The job is executed sequentially if due to not support parallel execution of jobs. Work stealing algorithm allows an idle processor execute task of another busy processor queue. Processor has no pending jobs in local deque and global EDF queue first job has a greatest dead line than other processors have at least one job at the top. Processor task should steal the earliest dead line from the topmost jobs on the other workers deque. Probably dead line based work stealing policy will positively increase the speedup of Parallel applications without affecting the schedulability of the other sequential jobs scheduled by global EDF.

Mainly Common method for real-time scheduling on Multiprocessor systems is to partition the tasks to be scheduled among the available processors. While simple to implement, this approach has several drawbacks when implementing .Further, partitioning is inflexible in that spare capacities cannot be reused on different processors. For these reasons, prior efforts on implementing servers on multiprocessors have focused on the usage of global scheduling algorithms. Two different types of global algorithms have been considered: job level fixed-priority algorithms and Pfair algorithms. In a job *level fixed-priority algorithm*, a job's priority, once assigned, does not change. Example for such an algorithm is global EDF. In contrast, *Pfair algorithms* break each job to be scheduled into quantum-length pieces of work, called *subtasks*, which are then scheduled. Pfair algorithms, in theory, have the advantage that a system's full capacity can be utilized: any sporadic real-time task system with total utilization at most m can be scheduled using Pfair algorithms with no deadline misses on an m-processor system [5]. In contrast, as with partitioning, caps on overall utilization must be used when using job level fixed-priority algorithms if every deadline must be met.

III. Proposed Approach

There Are No Boundaries for taking number of hard real time tasks in Extension of EDF- hl. Extension of this we call it as EDF-HSB, creates up to m non- migratory servers to execute Hard Real Time (HRT) tasks with zero tardiness. These servers are statically prioritized over the other servers in the system and can be provisioned independently of the periods of their clients. Each SRT task is serviced by a single server Queued BE jobs are scheduled by additional SRT servers. The SRT servers may miss their deadlines, but by bounded amounts only. M-CASH (10), which extends M-CBS by adding reclaiming techniques for reallocating processing capacity that, becomes available when a server completes early. In this work, global EDF is assumed to be the top-level scheduler. Solve the problem of integrating aperiodic jobs and HRT tasks by using Pfair-based algorithms as the top-level scheduler. To make best use of dynamic slack when servers complete execution early, EDF- HSB uses a spare capacity redistribution method similar to M-CASH [6]. Jobs that finish early donate their unused capacity to a global capacity queue. Both SRT tasks that are likely to be tardy and BE jobs can receive such capacities to improve performance.

IV. System Model

We consider the problem of scheduling a set τ of n fully pre-emptive, independent, sporadic real-time tasks concurrently with independent BE aperiodic jobs on a set of m identical multiprocessors with unit capacity. Each sporadic task $T_i = (e_i, p_i)$ is characterized by its worst-case *execution requirement*, e_i, its minimum inter-arrival time, or *period*, p_i, and its *utilization*, $u_i = e_i/p_i$ .Each such task generates a sequence of jobs $T_{i,j}$, where $j \geq 1$.We denote the instant that a job becomes ready for execution as $r_{i,j}$, and require $r_{i,j} + p_i \leq r_{i,j+1}$. If the jobs of a sporadic task T_i are always released p_i time units apart, starting at time 0, then T_i is called a *periodic* task. If a job $T_{i,j}$ of a sporadic task executes for $e_{i,j} < e_i$ time units, then the resulting unused capacity, $e_i - e_{i,j}$,is referred to as *dynamic slack*. If such a job does not receive an allocation of $e_{i,j}$ time units before its implicit deadline $d_{i,j} = r_{i,j} + p_i$,then it is *tardy*. Note that, if a job of a sporadic task is tardy, then the release time of the next job of that task is not delayed.All tasks and jobs are sequential, *i.e.*, the jobs of a sporadic task must execute in sequence, and each job can only execute on one processor at a time. We require all periods and deadlines to be some integral number of quanta (though execution costs can be non-integral). We assume this because actual hardware has inherent limits on timer resolution and reasonable overhead costs. For the purpose of the analysis, we consider preemption and migration costs to be negligible. However, we address these costs when discussing the issue of reclaiming spare capacities. EDF-HSB allows the system to be fully utilized and can flexibly deal with dynamic slack which meets the above requirements. The various mechanisms used in EDF-HSB are described in detail in later subsections. We begin with a general description of some of the underlying design choices.

EDF-HSB ensures the temporal correctness of HRT tasks by statically partitioning them among the available processors, and by encapsulating those assigned to each processor within a periodic *HRT server*, which executes only on that processor. These HRT servers offer three distinct advantages. **1.** Due to HRT tasks do not migrate, the analysis of their worst-case execution times is simplified. **2.** HRT servers require no over-provisioning, *i.e.*, such a server's utilization is simply the sum of the utilizations of its clients. **3.** HRT server's period can be sized independently of its client tasks, and thus its impact on SRT tasks and BE jobs can be adjusted freely. In contrast to our approach, in most (if not all) prior server approaches,

the server must be over-provisioned in order to avoid client deadline misses .In EDF-HSB, the top-level scheduler schedules the HRT servers just described along with the other tasks in the system. These other tasks are prioritized against each other using a global EDF policy. Hence, they may migrate among the processors in the system. In addition, these other tasks may experience bounded deadline tardiness. Such tasks include both the SRT tasks that are part of the system to be scheduled and also a collection of sporadic *BE servers*, which are responsible for scheduling BE jobs. We will use the term "non- HRT task" when collectively referring to the set of SRT tasks and BE servers. When we use the term "job" in reference to a non-HRT task, and that task is a BE server, we are referring to an invocation of the server as scheduled by the top-level scheduler, and not a job that the server itself schedules. Although EDF-HSB prioritizes HRT servers over other tasks, such servers may be considered ineligible for execution even when they have client tasks with unfinished jobs. The eligibility rules for these servers, given later, are defined so that non-HRT tasks can be given preference in scheduling when doing so would not cause HRT tasks to miss their deadlines.

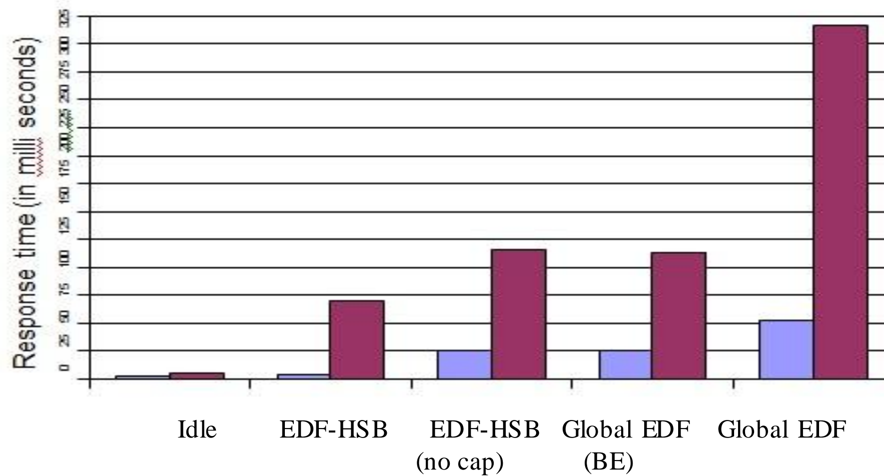
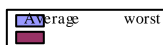


Fig 4: Average and worst-case Best Effort(BE) job response times under each tested scheme.



Arriving aperiodic jobs are placed in a single global FIFO queue. When a BE server is scheduled for execution, it services jobs from this queue until either the queue empties. We assume that there are a total of m BE servers, so that arriving BE jobs can be scheduled in parallel to the extent possible. If the system experiences intervals of under utilization, then the BE servers double as background servers. At runtime, the performance of the system is further enhanced through a novel use of spare capacity redistribution to lessen tardiness for non-HRT tasks and to adapt quickly to load changes. For example, even if one processor is fully committed to serving HRT tasks, EDF-HSB can still use dynamic slack released on that processor to improve the performance of non-HRT tasks. The redistribution of slack is controlled by a heuristic. By using different heuristics, it is possible to tune EDF-HSB for various task loads. Different heuristics can be defined depending on whether it is more important to lower BE job response times or to lower SRT task tardiness.

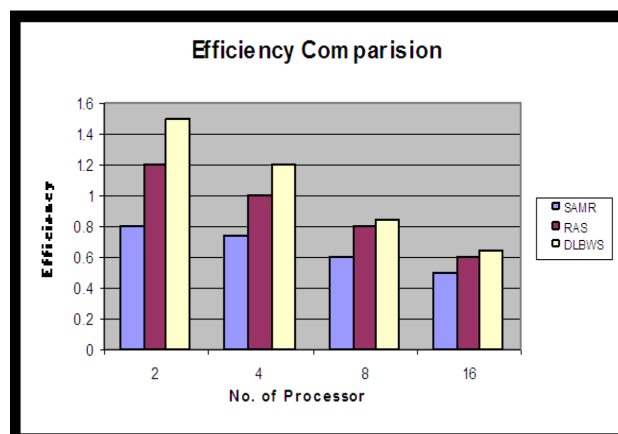


Fig 5: Comparison of Efficiency Vs Number of Processors

V. Conclusion And Future Scope

Support Dynamic Task Level Parallelism In Real Time Systems And Now Proposed Approach Is That Combines A Work Stealing Load Balancing Policy With Multicore Reservation Based Approach. Work Stealing Will Increase The Speedup Of Parallel Applications Without Affecting The Schedulability Of The Other Jobs Scheduled By EDF-HSB In Place Of Global EDF (8). Here Involve Of The Scheduling Of Periodic Tasks With Implicit Deadlines. It Was Claimed That U-EDF (9)Is Optimal For Periodic Tasks (I.E., It Can Meet All Deadlines Of Every Schedulable Task Set) And Extensive Simulations Showed A Drastic Improvement In The Number Of Task Preemptions And Migrations In Comparison To State-Of-The-Art Optimal Algorithms. However, There Was No Proof Of Its Optimality And U-EDF Was Not Designed To Schedule Sporadic Tasks.

Sreenath.M, received B.Tech degree in ICE from JNTU Hyderabad and M.Tech degree in Embedded Systems from JNTU Anantapur. Currently working as Assistant Professor in the Department of E.C.E., Annamacharya Institute of Technology and Sciences, Rajampet, Kadapa, AndhraPradesh, India. Areas of interests are embedded systems, Microprocessors & Microcontrollers and Control Systems.

Sukumar.P, received B.Tech degree from JNTU Hyderabad and M.Tech degree from ANU Guntur. Currently working as Assistant Professor in the Department of E.C.E., Annamacharya Institute of Technology and Sciences, Rajampet, Kadapa, Andhra Pradesh, India. Area of interests are, Microprocessors & Interfacing, embedded systems and Real Time Operating System.

Naganarasaiah Goud.K, received B.Tech degree in ECE from JNTU Hyderabad and M.Tech degree in Embedded Systems from JNTU Hyderabad. Currently working as Assistant Professor in the Department of E.C.E. , Annamachrya Institute of Technology and Sciences, Rajampet, Kadapa, Andhra Pradesh, India. Area of interests are embedded systems and Microprocessors & Microcontrollers

References

[1] Minakshi Tripathy and C.R. Tripathy, Centralized Dynamic Load Balancing Model for Shared Memory Clusters, Proceedings of the International Conference on Control, communication and Computing, Feb 18-20, 2010, pp 173-176.

[2] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in RTSS, 2001, pp. 193–202.

[3] **P. Berenbrink**, T. Friedetzky, and L. Goldberg. The natural work-stealing algorithm is stable. In Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS), pages 178–187, 2001.

[4] J. Dinan, S. Krishnamoorthy, D. B. Larkins,J. Nieplocha, and P. Sadayappan. Scioto: A framework for global-view task parallelism. In Proc.37th Intl. Conf. on Parallel Processing (ICPP), pages 586–593, 2008.

[5] A. Srinivasan and J.Anderson.Optimal rate based scheduling on multiprocessors. Journal of Computer and System Sciences, 72(6):1094–1117, September 2006.

[6] M. Caccamo,G. Buttazzo, and L. Sha. Capacity sharing for overrun control.In Proceedings of the 21st IEEE Real-Time Systems Symposium, pages 295–304, 2000.

[7] U. V. ¸Cataly¨urek, E. G. Boman, K. D. Devine,D. Bozdag, R. Heaphy, and L. A. Riesen.Hypergraph-based dynamic load balancing foradaptive scientific computations. In Proc. 21st Intl.Parallel and Distributed Processing Symposium (IPDPS), pages 1–11. IEEE, 2007.

[8] S. K. Baruah and T. P. Baker, "Schedulability analysis of global EDF," Real-Time Systems, vol. 38, no. 3, pp. 223–235, 2008.

[9] G. Nelissen, V. Berten, J. Goossens, and D. Milojevic, "Reducing preemptions and migrations in real-time multiprocessor scheduling algorithms by releasing the fairness," in RTCSA'11, 2011, pp. 15–24.

[10] R. Pellizzoni & M. Caccamo. The M-CASH resource reclaiming algorithm for identical multiprocessor platforms. Technical Report UIUCDCS-R-2006-2703, University of Illinois at Urbana-Champaign, 2006.

[11] Bill N. and Virginia L., Distributed Shared Memory: A Survey of Issues and Algorithms, JournalComputer - Distributed computing systems, Vol. 24,No. 8, August 1991, IEEE Computer Society Press.

[12] Kashif Bilal, Tassawar Iqbal, Asad Ali Safi and Nadeem Daudpota, Dynamic Load Balancing in PVM Using Intelligent Application, World academy of science, Engineering and Technology, Vol. 5, May 2005, pp 132-135.

[13] P. Sammulal and A. Vinaya Babu, Enhanced Communal Global, Local Memory Management for Effective Performance of Cluster Computing, IJCSNSInternational Journal of Computer Science and Network Security, Vol.8, No.6, June 2008, pp 209-215