A Knapsack Scheduling Algorithm for Soft Real-time Multiprocessor System

Shaohang Cui[1], Douglas Buchanan[2], and Ken Ferens[2]
[1]ERLPhase Power Technologies Ltd.
[2]Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada
scui@erlphase.com, Douglas.Buchanan@ad.umanitoba.ca, Ken.Ferens@ad.umanitoba.ca

Abstract—Due to the development of commercial multicore architectures, scheduling in Multiprocessor real-time system has attracted huge research interests and various algorithms have been proposed. However, most prior research on real-time scheduling on multiprocessors has focused only on hard real-time systems where no deadline may ever be missed. Due to the increasing need of soft real-time multiprocessor systems where deadline violation is tolerable, scheduling in overloaded soft real-time systems should be studied as well. In this paper, we consider the time/utility functions (TUFs) express the utility of completing a task as a function of that task's completion time. Our objective is to maximize the overall system utility with given computation capability constraints and TUFs of tasks. We first formulate the scheduling problem to an allocation optimization problem. Then, for practical implementation, we propose a knapsack method to achieve the suboptimal solution. Simulation results demonstrate that the proposed knapsack method could achieve about 90% of the optimized performance.

Keywords - Soft-real time scheduling; knapsack greedy scheduling.

I. INTRODUCTION

Real-time multiprocessor systems are now ubiquitous. The need for multiprocessor real-time analysis has been well realized in the past 10 years due to the development of commercial multicore architectures and the research conducted on the design of effective scheduling algorithms. The difficulty with achieving effective scheduling algorithms in multiprocessor systems is that the algorithm must optimally allocate processors to tasks, and then optimally schedule the tasks in each processor, which is a doubly complex endevour [1].

Research on real-time scheduling has largely focused on recurring tasks, i.e., for each task, a potentially infinite sequence of events (or jobs) arrive gradually [2], [3], [4]. These algorithms and systems under consideration could be classified to different categories for various perspectives. From the perspective of processors, multiprocessor systems can be classified into [5]: (1) Heterogeneous where the processors are different; (2) Homogeneous where the processors are identical (3) Uniform where processors operate at different integral multiple of a unit speed. From the perspective of the degree of run-time migration, algorithms could be classified to [6]: (1) no migration, where tasks are partitioned and allocated to different processors before their running (offline), and uniprocessor scheduling algorithm is applied to each processor; (2) full migration, where jobs are allowed to arbitrarily migrate across processors during their execution; and (3) restricted migration, where some form of migration is allowed, e.g., in a job's scale.

However, most prior research on real-time scheduling on multiprocessors has focused only on hard real-time systems where no deadline may ever be missed, which also indicated that the systems under analysis must be under-loaded for a feasible algorithm design. When resource overloads occur, meeting deadlines of all activities is impossible as the demand exceeds the supply. For a soft real-time, violation of the deadline constraints results in degraded quality, but the system can tolerate such violation and continue to operate [7]. Due to the increasing need of soft real-time multiprocessor systems with large computation requirements, such as tracking, signal-processing, and multimedia systems, scheduling in overloaded soft real-time systems should be studied as well.

In a soft real-time system, deadlines by themselves cannot express both urgency and importance, e.g., the most urgent activity can be the least important, or vice versa [8]. In this case, in order to consider the importance and urgency of tasks jointly, we consider the abstraction of time/utility functions (or TUFs) as similar in [9] that express the utility of completing a job as a function of that job's completion time. Utility is a task-specific and time-related measurement of the value to the system. TUF quantifies the utility of completing each task at a given time. We consider a simple monotonically non-increasing functions as TUFs that, for a given task, once a job of this task arrives and is ready to process, the more time used to complete the job, the smaller its value to the system. Our objective is to maximize the

overall system utility with given computation capability constraints and TUFs of tasks. For simplicity, we consider a homogeneous multiprocessor system with full migration capability and worst case demand.

In [10][11], it has been proved that for a homogeneous multiprocessor system with full migration capability, as long as the utilization requirements of all tasks is less than the system computation capability, a scheduling policy could be achieved by the proposed LLREF algorithm. Our idea for the soft real-time system is that, based on the system computation capability, we are going to find an allocation of utilization of processors to each tasks to maximize the overall system utility. Once the allocation problem is solved, a scheduling policy could be achieved based on LLREF algorithm. In this way, we convert TUFs to utilization/utility functions (UUF), and formulate the allocation problem as an optimization problem. Due to practical reasons, we then propose a knapsack method to achieve a suboptimal solution for the allocation problem.

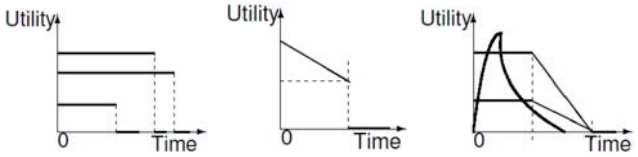
The rest of this paper is organized as follows. In Section 2, we propose the system model and formulate the scheduling problem to an allocation optimization problem. Then, for practical applications, we propose a knapsack method to achieve the suboptimal solution in Section 3. The performance evaluation of the concerned algorithms is presented in Section 4. And Section 5 concludes with a summary of the work presented in this paper.

II. SYSTEM MODEL

Similar to [10], we consider a homogeneous multiprocessor system of M identical processors. The application under consideration consist of a set of tasks, denoted by $T = \{T_1, T_2, ..., T_N\}$. Each task T_i has a number of jobs, and these jobs may be released either periodically or sporadically. All tasks are assumed to be independent, i.e., they do not share any resource or have any precedence. Full migration is allowed, i.e., tasks may be pre-empted at any time. As it is a soft real-time system, it is acceptable for a job of a task to be completed after the arrival of the next job. The constraint is that all tasks could be completed steadily, say, the overall process rate should be no smaller than the overall demands. Worst case demand is considered here, say, there is a given minimum time e_i required to complete a task i in the worst case.

A. TUFs and UUFs

On the one hand, a hard real-time system will result in total system failure if a hard deadline of any job is missed, and there can be no benefit in completing any job later than its hard deadline. A hard real-time system results in such failure when resource demand exceeds system supply, or, in other words, when resource-overload occurs. On the other hand, a soft real-time system may continue to perform tolerably if any job misses its deadline, albeit with degraded performance. In other words, completing an activity at any time in a soft real-time system, even after its deadline, will result in some utility (benefit) to the system, although that benefit (utility) depends on (is proportional to) the activity's completion time. Generally speaking, the amount of benefit of completing a task after its deadline decreases with increasing actual time of completion. Many systems quantify the amount of benefit of completing a task after its original deadline using the so called time-utility function [8], which allows the semantics of soft real-time constraints to be precisely specified. TUF quantifies the utility (benefit) of completing a job as a function of time, including before and after that job's deadline time. Some examples of TUFs are shown as in Fig. 1.



(a) (b) (c)

Fig. 1 Example TUF Time Constraints. (a) Step TUFs; (b) MITRE/TOG AWACS TUF [12]; (c) GD/CMU BM/C2 TUFs [13].

This paper defines optimal scheduling of tasks to processors by the maximization of the so called Utility Accrual (UA) function [8]. The UA function is the accrued benefit of the activities completing their tasks at certain times, given that the utility of the tasks are described using the time-utility (time-benefit) functions. For example, the accrual method may be a simple sum, so that an optimal schedule is obtained when the sum of each activity's attained benefit is maximized. For simplicity, this paper considers monotonically non-increasing time-utility functions, so that, for a given task, once a job of this task arrives and is ready to process, the more time used to complete the job, the smaller is its benefit to the system. Given the system's resource capability, this paper's objective is to maximize the overall system utility, given the task time-utility functions.

In [11], a scheduling algorithm called LRE-TL was proposed for homogeneous hard real-time multiprocessor system with periodic or sporadic arriving jobs, in which it was proved that as long as the overall utilization requirements of all tasks is less than the processing capability, a feasible scheduling policy could be achieved to meet all task deadlines. The utilization of a task is defined as $u_i = \frac{e_i}{\min\ (P_i, D_i)}$, where e_i is the longest execution time of a job of a task i, P_i is the smallest interval between any of task i's two successive jobs, and D_i is task i's deadline.

For the soft real-time multiprocessor system under consideration, if we could find a time interval d_i for each task i, which is the maximum allowed interval between task i's arrival and completeness, we could get a utilization: $u_i = \frac{e_i}{d_i}$. Based on LRE-TL algorithm, as long as $0 \le u_i \le 1$ and $\sum_{i \in N} u_i \le M$, where M is the number of processors, a feasible scheduling policy could be acquired. Thus, the

scheduling problem could be converted to an optimization problem: find d_i to maximize the overall utility given the constraints for u_i. Or TUF could be converted to utilization utility functions (UUFs) which describe the relationship between a task's utility and utilization. In this way, the optimization problem is an allocation problem: how to allocate utilization to different tasks so as to maximize the overall utility, given the overall available utilization (processing ability) and UUFs.

B. Optimization Problem

From the above analysis, we could formulate the scheduling problem in a soft real-time multiprocessor system as an optimization problem for allocation of utilization to tasks as follows:

$$max_{u_i} \sum_i U_i(u_i), \text{s.t.}, \sum_i u_i \le \mathrm{M}, 0 \le u_i \le 1 \qquad (1)$$

When $U_i(u_i)$ are convex functions on the constrained set, it is a convex programming problem and could be easily solved by calculating the KKT conditions, i.e., let:

$$\mathrm{L(A,u)} = \sum_i U_i(u_i) + a_0\left(\sum_i u_i - M\right) - \sum_i a_i u_i + \sum_i a_{N+i}(u_i - 1), \text{where } A = [a_0, \ldots, a_{2N}] \qquad (2)$$

From KKT conditions, we have:

$$\frac{\mathrm{dL}}{\mathrm{du}_i} = 0 \qquad (3)$$

$$a_0\left(\sum_i u_i - M\right) = 0 \qquad (4)$$

$$\sum_i a_i u_i = 0 \qquad (5)$$

$$\sum_i a_{N+i}(u_i - 1) = 0 \qquad (6)$$

Note that to solve KKT conditions, U_i should be differentiable on the interval [0,1].

III. KNAPSACK GREEDY ALGORITHM

In practice, sometimes it may be not easy to acquire the analytic expression of the TUF and UUF, or it is too complex to solve the convex programming problem, or even U_i may be not be convex nor differentiable on [0,1]. In this case, we may use a Knapsack method to achieve the suboptimal solution.

Intuitively, to maximize the overall utility (benefit), higher utilization should be allocated to tasks which could achieve high utility with small utilization first. However, as the relationship between utilization and utility of a task is determined by its UUF, which is not a constant, we need to find a suitable point which is most profitable, say, how much utilization should be allocated to a task to achieve utility.

Based on this consideration, we divided the UUF to 9 zones as shown in Fig. 2.

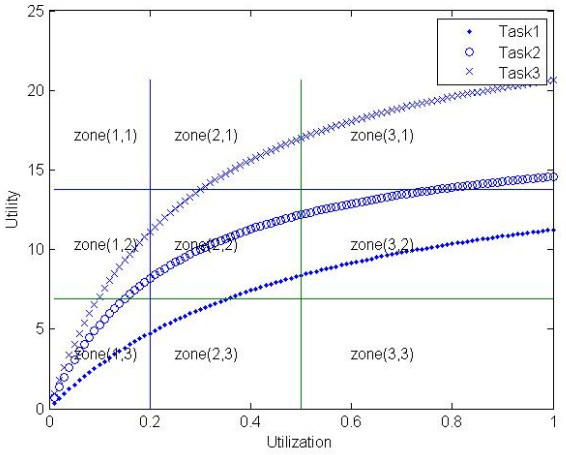


Fig. 2 UUFs and zones.

We select UUF $U_i = A_i - B_i/(u_i + B_i/A_i)$ where A_i and B_i are task specific constant as the example. u=0.2,0.5 and $U = \frac{1}{3}Y, \frac{2}{3}Y$. The Y is the maximum achievable utility of all tasks, given the maximum utilization (utilization = 1). In Fig. 2, Y ≈ 20.64 is a random number.

Each zone is abstracted as represented by its centroid, which is related with a required utilization and an estimated utility, as shown in Table 1.

Table 1 Required utilization and estimated utility for zones.

Zone(i,j)	**i=1**	**2**	**2**
j=1	(0.1, 5/6 Y)	(0.35, 5/6 Y)	(0.75, 5/6 Y)
2	(0.1, 1/2 Y)	(0.35, 1/2 Y)	(0.75, 1/2 Y)
3	(0.1, 1/6 Y)	(0.35, 1/6 Y)	(0.75, 1/6 Y)

For a task passing a zone, by estimation, within this zone, this task could achieve the utility with a utilization determined by the centroid of this zone. Based on its UUF, a task will pass several zones. For a task passing zones in the figure above, from left to right, more utilization will be allocated to a task. From low to high, more utility could be achieved by a task. A task passing several zones means it could be processed with several policies, i.e., requested utilization and achievable utility. In this way, these tasks' UUF have been fuzzed as each task many have several allocation policies. Now, the problem is, given all the tasks and related policies, how to maximize the overall utility.

This problem could be converted to a group Knapsack problem. However, given the requested utilization in each zone (the weight in Knapsack problem) is decimal, it is not efficient to solve it by using Knapsack methods. A greedy method is then introduced, which consists of three steps:

1. For a zone, its profit rate is the estimated utility divided by the required utilization. A task will select a zone with the highest profit rate from all the zones it passes, and use the corresponding allocation policy. The task with the highest profit rate will be allocated with its required utilization first. In this way, Task2 and Task3 will be allocated based on the policy for zone (1,2), and Task1 will be allocated based on the policy for zone (2,2).

2. After all tasks have been allocated with their required utilization to achieve the most profitable utility, if there is still abundant resource, tasks which could increases its utility markedly, should be provided with supplementary utilization, say, which could go from a lower zone to a higher zone by increasing utilization.

3. After tasks cannot achieve marked utility improvement by increasing utilization, if there is still abundant resource, such resource will be provided to all tasks equally, given each task could get utilization 1 mostly.

In this way, based on Table 3, if there is abundant resources remaining, task3 will be allocated first by increasing its utilization to 0.35, and task2 will then be allocated by increasing its utilization to 0.75. Note that based on step 1, if a task is already in zone(i',j), we do not need to take the number of marked changes happen in $i \le i'$. If a task passes several zones for the same i, only a higher zone will be kept.

Table 2 Task related zones zone(i,j).

Task1	Zone (1,3)	Zone (2,2)	Zone (3,2)
Task2	Zone (1,2)	Zone (2,2)	Zone (3,1)
Task3	Zone (1,2)	Zone (2,1)	Zone (3,1)

Table 3 Marked increase as utilization increases.

Task1	Zone (1,3)	Zone (2,2)	Zone (3,2)
Task2	Zone (1,2)	Zone (2,2)	Zone (3,1)
Task3	Zone (1,2)	Zone (2,1)	Zone (3,1)

IV. PERFORMANCE

Since the optimization problem could only be solved by KKT conditions when $U_i(u)$ are differentiable and convex functions on the constrained set $u \in [0,1]$. To compare the performance of the proposed knapsack algorithm to the optimal solution, we select the UUF as $U_i = A_i - B_i/(u_i + B_i/A_i)$, where A_i is a random number uniformly distributed on [5,20], and B_i is a random number generated by normal distribution norm (5,5).

For N=10, increase M for 1 to 10, we acquired the performance of knapsack algorithm and optimal solution as shown in Fig. 3. Note that performance of both algorithms has been normalized by the system demands. From this figure, we could see that the proposed knapsack algorithm could achieve about 90% performance of the optimal solution even when M is small. As M increases and the system becomes less overloaded, the knapsack algorithm could achieve more utility until satisfy all system demands.

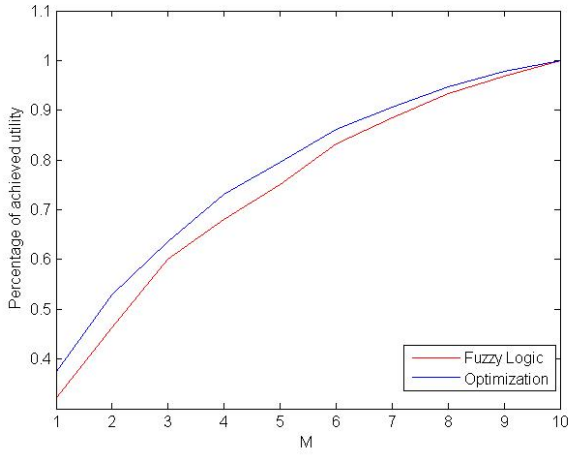
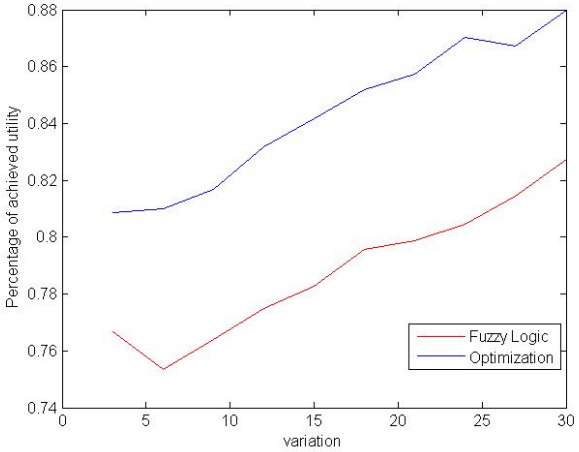
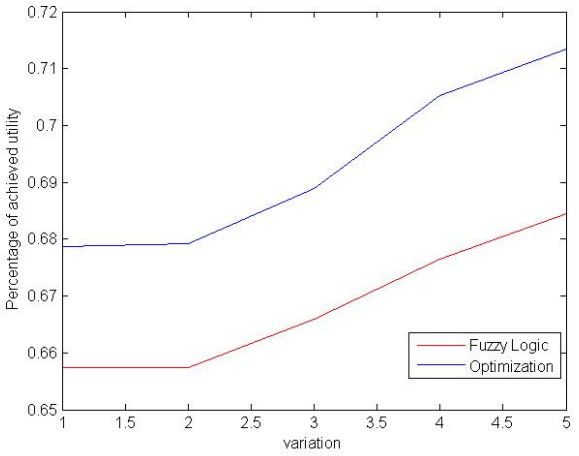


Fig. 3 Performance vs. processors (M).



(a)



(b)

Fig. 4 Performance vs. variation. (a) variation of A_i; (b) variation of B_i.

Select M=3. Set A_i a random number generated by abs(norm(30,v)), and B_i is a random number generated by normal distribution norm(5,5). Increasing v from 3 to 30 we got a performance result as shown in Fig. 4(a). We could see that in general, as the variation, or the difference of tasks increases, performance of both knapsack algorithm and optimization algorithm will increase.

V. CONCLUSIONS

When resource overloads occur, meeting deadlines of all activities is impossible as the demand exceeds the supply, while it is still acceptable in a soft real-time system. In this paper, we consider the time/utility functions (TUFs) express the utility of completing a task as a function of that task's completion time. Our objective is to maximize the overall system utility with given computation capability constraints and TUFs of tasks. We first formulate the scheduling problem to an allocation optimization problem. Then, for practical implementation, we propose a knapsack method to achieve the suboptimal solution. Simulation results demonstrate that the proposed knapsack method could achieve about 90% of the optimized performance.

REFERENCES

[1] O. Zapata and P. Mejia-Alvarez. Analysis of real-time multiprocessors scheduling algorithms. In Proceedings of the Real-Time Systems Symposium (RTSS), 2003.

[2] Baker, T.P., "An analysis of EDF schedulability on a multiprocessor," Parallel and Distributed Systems, IEEE Transactions on , vol.16, no.8, pp.760-768, Aug. 2005.

[3] Anderson, J.H.; Srinivasan, A., "Early-release fair scheduling," Real-Time Systems, 2000. Euromicro RTS 2000. 12th Euromicro Conference on , vol., no., pp.35-43, 2000.

[4] Jinkyu Lee; Easwaran, A.; Insik Shin, "LLF Schedulability Analysis on Multiprocessor Platforms," Real-Time Systems Symposium (RTSS), 2010 IEEE 31st , vol., no., pp.25-36, Nov. 30 2010-Dec. 3 2010.

[5] Davis, Robert I. and Burns, Alan., "A survey of hard real-time scheduling for multiprocessor systems", Journal of ACM Comput. Surv., vol. 43, no. 4, pp35-34, 2011.

[6] Carpenter, J.; Funk, S.; Holman, P.; Srinivasan, A; Anderson, J;Baruah, S., "A categorization of real-time multiprocessor scheduling problems and algorithms," Handbook On Scheduling Algorithms, Methods, And Models, 2004.

[7] Muppala, J.K.; Woolet, S.P.; Trivedi, K.S., "Real-time systems performance in the presence of failures," Computer, vol.24, no.5, pp.37-47, May 1991

[8] Wu, H.; Ravindran, B.; Jensen, E.D.; Peng Li, "Time/utility function decomposition techniques for utility accrual scheduling algorithms in real-time distributed systems," Computers, IEEE Transactions on , vol.54, no.9, pp.1138,1153, Sept. 2005.

[9] Ravindran, B.; Jensen, E.D.; Peng Li, "On recent advances in time/utility function real-time scheduling and resource management," Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on, vol., no., pp.55-60, 18-20 May 2005.

[10] Hyeonjoong Cho; Ravindran, B.; Jensen, E.D., "An Optimal Real-Time Scheduling Algorithm for Multiprocessors," Real-Time Systems Symposium, 2006. RTSS '06. 27th IEEE International, vol., no., pp.101,110, Dec. 2006.

[11] Shelby Funk, "LRE-TL: an optimal multiprocessor algorithm for sporadic task sets with unconstrained deadlines," Real-Time Systems, vol. 46, no. 3, pp 332-359, 2010.

[12] R. Clark, E. D. Jensen, A. Kanevsky, J. Maurer, P. Wallace, T. Wheeler, Y. Zhang, D. Wells, T. Lawrence, and P. Hurley, "An Adaptive, Distributed Airborne Tracking System," in Proceedings of The IEEE Workshop on Parallel and Distributed Systems, ser. LNCS, vol. 1586. Springer-Verlag, pp. 353-362, April 1999.

[13] D. P. Maynard, S. E. Shipman, R. K. Clark, J. D. Northcutt, R. B. Kegley, B. A. Zimmerman, and P. J. Keleher, "An Example Real-Time Command, Control, and Battle Management Application for Alpha," Department of Computer Science, Carnegie Mellon University, Archons Project TR-88121, December 1988.