A Brief Survey of Multi-Processor Scheduling For Hard Real-Time Systems

Xin Lin[a], Xiaorong Zhu[a], Lijia Liu[a]

[a]*Department of Computer Science, The University of Texas at Austin*

Abstract

In class, both of scheduling algorithms [1] and priority inheritance protocols [2] in the context of a single processor were examined in details. Nevertheless, the emergence and popularity of distributed computing system gave rise to the need to solve multi-processor scheduling and priority inheritance problems. As the supplementary study, this paper surveys existing scheduling algorithms in the context of multiple processors. The very first section outlines the background of multi-processor scheduling problems, as well as system models, terminology, and the metrics of scheduling algorithms. After that, partitioned scheduling and global scheduling, as the primary objects of our research, will be fully explored. Lastly, we conclude our exploration in the topic of scheduling algorithm.

Keywords: System, Scheduling Algorithm, Task Management

1. Introduction

Real-time systems have become prevalent in diverse application areas, such as traffic controls, embedded automotive electronics, networked multimedia etc. Typically, a real-time system comprises a real-time computer system and a controlled object, which are connected through sensors, actuators and some other input-output interfaces. The controlling real-time computer reacts/responds to its controlled object based on the currently known information about the object. For example, a real-time computer controls a device according to the data read by sensors at periodic intervals and responds by sending signals to actuators within a time bound.

The purpose of real-time systems is to meet various timing constraints imposed on them by its external world during the operation of the system.

The instant at which a response from the controlling system must be delivered to its controlled system is called a deadline. A deadline can be classified into three categories[3]:

- Soft Deadline: the response still has effectiveness even after the deadline has passed. However, the later the execution of a task after the deadline, the more penalty it pays.
- Firm Deadline: the response has no utilize if the deadline has passed. Moreover, the earlier the execution of a task before a firm deadline, the more rewards it gains.
- Hard Deadline: a task must complete its computation by the deadline, otherwise a catastrophe could happen.

Commands and Control systems, Air traffic control systems are examples for hard real-time systems. On-line transaction systems, airline reservation systems are soft real-time systems[4]. Such a system is called embedded real-time system if it is a component of a larger system.

Companies are motivated to build embedded real-time systems by their advanced effectiveness and flexibility. In order to beat the competition in industries, they need to satisfy customers needs and lower associated costs to the maximum extent possible. Such aspiration has resulted in a rapid progress in the area of embedded real-time technology. For instance, silicon vendors used to increase processor performance by concentrating on the miniaturization needed. However, this approach has led to problems with both high power consumption and excessive heat dissipation[4]. Thus using multiprocessor platforms for high-end real-time applications is an increasing trend toward instead[4].

Multiprocessor systems can be classified into three categories from the perspectives of scheduling[4]:

- Heterogeneous: processors in the system are different. Not all tasks may be able to execute on all processors and the execution rate of a task depends on both the processor and the task.
- Homogeneous: processors in the system are identical. The execution rate of all tasks is the same on all processors. Thus it depends only on the task.

- Uniform: the execution rate of a task depends only on the speed of processors. For example, a processor with speed 2 will execute all tasks at exactly twice the rate of a processor with speed 1.

The paper is organized as follows: section 2 reviews the foundational concepts in multiprocessor real-time scheduling algorithm. Section 3 and section 4 discuss the partitioned scheduling and global scheduling in detail, respectively. A conclusion is given at the end of this paper.

2. Preliminaries

This section reviews the fundamental terminologies and notations used in multiprocessor scheduling algorithms.

2.1. Multiprocessor Scheduling

In computer science, multiprocessor scheduling is an NP-hard optimization problem[wiki]. The problem can be described as "given a set J of jobs where job j_i has length l_i and a number of processors m, what is the minimum possible time required to schedule all jobs in J on m processors such that none overlap?"[5]. Clearly multiprocessor schedulers should concern with the allocation of the resources and the satisfaction of timing constraints[4]:

- Resource Allocation: on which processor a task executes.
- Propriety Determination: when and in what order, with respect to jobs of other tasks, each job executes so that all jobs can be finished before their deadlines.

Then real-time scheduling algorithms for multiprocessor systems can be classified from perspectives of allocation changes and priority changes (referred to as migration-based and priority-based classifications [6]).

2.2. Classification of Real-time Scheduling Algorithms

Real-Time scheduling can be categorized into hard scheduling and soft scheduling based on the types of deadlines. The hard real-time scheduling can be further classified into two types: static and dynamic. A static scheduling generates its scheduling decisions off-line according to the prior information of the jobs in a task. The decisions are made at compile time, thus its run-time overhead is small. By contrast, the scheduling decisions are

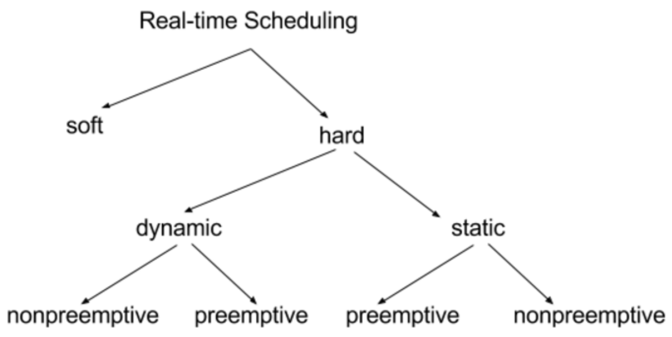

Figure 1: Taxonomy of Real-time Scheduling Algorithms

made at run time in dynamic scheduling system, which leads to significant overheads. Furthermore, both dynamic scheduling and static scheduling can have preemptive or nonpreemptive scheduling:

- Preemptive: the currently executing task can be preempted by a higher priority task at anytime
- Nonpreemptive: the currently executing task can not be preempted until completion

In this paper, we are mainly concerned with dynamic preemptive scheduling algorithms. Figure 1 illustrates a rough taxonomy of real-time scheduling algorithms.

More specifically, the dynamic scheduling algorithm can be classified by its priority policy[4]:

- Fixed task-level priority: the jobs of a task have a single fixed priority, such as Rate Monotonic RM algorithm
- Fixed job-level priority: the jobs of a task may have different priorities, but each job has a single static priority, such earliest deadline first (EDF) scheduling.
- Dynamic priority: A single job may have different priorities at different times, such as least laxity first (LLF) scheduling.

From the respective of allocation policy, the real-time scheduling algorithms for multiprocessor systems can be also classified as follows[4]:

- Non-migration: the jobs of a task must execute on one single processor and migration is not permitted.
- Task-level migration: the jobs of a task may be allocated to different processors, however, each job can only execute on one single processor.
- Job-level migration: a single job can be executed on different processors, however, no parallel execution of a job is permitted.

Scheduling algorithms without migrations are referred partitioned algorithms while those that permits migrations are referred as global algorithms. These two algorithms have been discussed in detail in section 3 and section 4, respectively.

2.3. Classifications of Task Models

Two simple task models have been defined for multiprocessor real-time scheduling[4]:

- periodic task model: the jobs of a task arrive strictly periodically, separated by a fixed time interval.
- sporadic task Model: the interval time of two continuous jobs from a task must be larger than a fixed time interval.

Each task π_i is characterized by: the deadline D_i, the period/minimum separation time P_i, the worst-case computation time C_i and the utilization U_i, where $U_i = C_i/P_i$. The response time R_i of a task is defined as the interval between the task request time and the task satisfaction time. The hyperperiod $H(\pi)$ is defined as the least common multiple (lcm) of all the periodic tasks.

Some other important terminologies realted to taskset are[4]:

- schedulable task: its worst-case response time under a scheduling algorithm is less than or equal to its deadline.
- schedulable taskset: all of its tasks are schedulable under a scheduling algorithm.
- feasible taskset: for each possible sequence of jobs that may be generated by a taskset in given a system, there exists some scheduling algorithm to satisfy it without missing any deadlines.

2.4. Notations

The notations that are used frequently in multiprocessing real-time scheduling and in this paper are as follows[4]:

- π_i: task i at priority level i
- B_i: blocking time at priority level i
- C_i: worst-case execution time of task π_i
- D_i: relative deadline of task π_i
- δ_i: density of task pi_i , $\delta_i = C_i/min(D_i, T_i)$
- δ_{max}: maximum density of any task in the taskset
- δ_{sum}: taskset density (sum of task densities)
- f_A: speedup factor (resource augmentation factor) for scheduling algorithm A
- $h(t)$: processor demand in the interval $[0, t)$
- $H(\pi)$: hyperperiod of the taskset
- $load(\pi)$: processor load of taskset π
- $load(\pi, k)$: processor load of taskset π , due to tasks of priority higher than or equal to k
- m: number of processors
- $M_A(\pi)$: minimum number of processors needed to schedule taskset π using scheduling algorithm A
- n: number of tasks
- N: number of jobs (typically in the hyperperiod of the taskset)
- R_i: worst-case response time of task π
- R_A: approximation ratio for scheduling algorithm A
- t: time

- T_i: minimum interarrival time of task π
- u_i: utilization of task π
- u_{max}: maximum utilization of any task in the taskset
- u_{sum}: taskset utilization
- U_A: utilization upper bound for scheduling algorithm A

3. Partitioned Scheduling

In this section, we will review some partitioned approaches to multiprocessor real-time scheduling, and then compare their performance.

3.1. Characteristic of Partitioned Scheduling

Partitioned scheduling provides capability for performing parallel processing and also for automation of batch execution of multiple processes. It can improve performance by breaking down a process that works on a large data set, to multiple parallel processes that work on smaller data sets, each contains part of the original data set. The general solution involves two algorithms: one to assign tasks to processors, known as the allocation, the other to schedule tasks that are assigned to each individual processor. The partitioning strategy also requires that all occurrences of a task to be executed on the same processor.

One main advantage of partitioned scheduling is that, after the allocation of tasks to processors, we can apply the optimal real-time scheduling techniques and analyses for uni-processor systems to each individual processor.

Comparing to the global scheduling, partitioned scheduling also has some advantages.

- A task that overruns its worst-case execution time will not affect tasks on other processors.
- There is no migration cost as each task only runs on a single processor.
- For each processor, the run-queue is much smaller comparing to the single global queue in the global scheduling, thus the overheads of managing the run-queue can be neglected.

However, the main disadvantage of the partitioned approach is that the allocation problem is analogous to a NP-Hard problem.

3.2. Rate Monotonic Next Fit Scheduling (RMNF)

The rate-monotonic-next-fit scheduling [7] is a partitioned scheduling algorithm for multiprocessors systems. According to this algorithm, tasks are first sorted in the descending order of their request rates. The scheduling scheme is that:

1. Starts from the first task i, and the first processor j.
2. Assign the task i to the processor j if j meets the requirement that together with all tasks that have been assigned to the processor j, tasks can be feasibly scheduled on processor j according to the rate-monotonic scheduling algorithm for a single processor. Otherwise, assign i to j + 1 and increase j by 1.
3. Get the next task and repeat the previous two steps, unless there is no task left.

Let N be the number of processors required to be feasibly schedule a set of task by the RMNF algorithm, and N_0 be the minimum number of processors required to feasibly schedule the same set of tasks. Then as N_0 approaches infinity, we can get that [7]

$$2.4 \le \frac{N}{N_0} \le 2.67 \tag{1}$$

3.3. Rate Monotonic First-Fit Scheduling (RMFF)

The rate-monotonic-first-fit scheduling algorithm [7] is similar to the RMNF. According to the RMFF, the tasks are first sorted in the descending order of their request rates. We will use a counter N to be the number of processors required for scheduling the given tasks. N is first initiated to 1. The scheduling scheme is that:

1. Starts from the first task i.
2. Starts from the lowest-indexed processor. Assign the task i to the first processor that meets the requirement that together with all tasks that have been assigned to the processor, tasks can be feasibly scheduled on the processor according to the rate-monotonic scheduling algorithm for a single processor. If no processor with index less than N meets the requirement, increase N by 1.
3. Get the next task and repeat the previous step, unless there is no task left.

Similarly, the bound of the rate of the number of processors N required to be feasibly schedule the task set by this algorithm, and the minimum number of processors N_0 required to be feasibly schedule the same set of tasks can be obtained when N_0 approaches infinity:

$$2 \leq \lim_{N_0 \to \infty} \frac{N}{N_0} \leq \frac{4 \times 2^{1/3}}{1 + 2^{1/3}} \approx 2.33 \qquad (2)$$

3.4. Rate Monotonic Best-Fit Scheduling (RMBF)

The rate-monotonic-best-fit scheduling [8] algorithm is based on the bin-packing heuristic. Best-Fit scheme chooses to assign tasks on a processor that can maximize the utilization of that processor.

Similar to RMNF and RMFF, we first sort all given tasks according to non-decreasing periods. Keep a counter N to be the number of processors required for scheduling the given set of tasks. N is set to 1 at the beginning. The scheduling algorithm works like this:

1. Starts from the first task i.
2. Starts from the lowest-indexed processor j. Let k_j and U_j denote the number of tasks already assigned to the processor j and the total utilization of the k_j tasks. Let u_i denote the utilization of the task i. Find the smallest-indexed processor j such that task i together with all k_j tasks can be feasibly scheduled by the rate-monotonic scheduling algorithm, and $2(1 + \frac{U_j}{k_j})^{-k_j} - 1$ be as small as possible, then assign task i to j, and set $k_j = k_j + 1$, $U_j = u_j + U_j$. If $j < m$ then set $m = j$.
3. Get the next task and repeat the previous step, unless there is no task left.

Similarly, the bound of rate of N and N_0 (with the same definitions in RMFF) can be obtained when N_0 approaches infinity:

$$2.3 \leq \lim_{N_0 \to \infty} \frac{N}{N_0} \leq 2 + \frac{3 - 2^{3/2}}{2(2^{1/3} - 1)} \approx 2.33 \qquad (3)$$

3.5. Comparison and Summary

In this section, we had introduced the three partitioned scheduling algorithms RMNF, RMFF and RMBF. All these three algorithms require that tasks are sorted according to their non-decreasing periods/request rates. They are only applicable to be applied to the situations that the periods

Table 1: Approximation Ratios

Algorithm	**Approximation Ratio** $\frac{N}{N_0}$
RMNF	2.67
RMFF	2.33
RMBF	2.33

of incoming tasks are fixed and static. The performance of these three algorithms are evaluated in the approximation ratio, which is defined to be the rate of N and N_0, the number of processors required to be feasibly schedule a set of task by the algorithm and the minimum number of processors required to feasibly schedule the same set of tasks. As shown in the Table 1, RMNF has a approximation ratio as 2.67, while RMFF and RMBF have better performance.

4. Global Scheduling

In this section, we will dive into another branch of scheduling strategies – global approaches to multiprocessor real-time scheduling.

4.1. Overview

Global scheduling algorithms, as its name suggests, globally schedules any feasible periodic task set. In contrast to partitioned scheduling, global scheduling schedules jobs and tasks in one single shared queue instead of multiple local, dedicated queues. By this means, the automatic load balancing and lower average response time can be achieved by global approaches. In addition to this, global approaches also take advantages of the simpler implementations and the existence of optimal schedulers.

Global scheduling startegies include Global Fixed-Job-Priority Scheduling, Global Fixed-Task-Priority Scheduling, and Global Dynamic Priority Scheduling. Although there are various categories of global scheduling algorithms, the focus of this paper is on the Global Dynamic Priority Scheduling. In the following subsection, it will be chacterized in details.

4.2. Global Dynamic Priority Scheduling

In this subsection, we will present our in-depth exploration to the track of global dynamic priority scheduling algorithm. To the best of our knowledge, a number of global dynamic priority scheduling algorithms are optimal for periodic tasksets with explicit or implicit deadlines. For example, Proportionate Fairness algorithm and its variants including PD, PD^2, ERFair, BF, SA [9], and LLREF [10] as well, are all optimal for offline environment. Nevertheless, no algorithms until now are optimal to cope with online preemptive scheduling problem, where tasksets are sporadic and multi-processor environments are enforced. On the other hand, despite of its optimality and dominance in theory, the usage of global dynamic priority algorithms are limited in practice. This is because the existence of frequent preemption and migration between tasks gives rise to excessive overheads in potential.

The following part of this subsection will provide brief summary of three classic global dynamic priority scheduling algorithms. They are respectively Proportionate Fairness Algorithm (PFair), and Largest Local Remaining Execution First (LLREF).

4.2.1. PFair

Baruah et al [11] introduced Proportionate Fairness Algorithm. The Pfair class of algorithms that allow full migration and fully dynamic priorities have been shown to be theoretically optimal – i.e., they achieve a schedulable utilization bound (below which all tasks meet their deadlines) that equals the total capacity of all processors. Here are some fundamental principles of Proportionate Fairness algorithms:

1. Timeline is divided into equal length slots.
2. Task period and execution time are multiples of the slot size.
3. Each task receives amount of slots proportional to its task utilization.

The essential part of PFair algorithms is the quantum-based optimization defined over the lag of each task $lag(\tau_i, t)$, with the goal of minimizing the maximum lags of all tasks $\max_t |lag(\tau_i, t)|$.

$$\underbrace{lag(\tau_i, t)}_{error} = \underbrace{t \cdot (\frac{c_i}{T_i})}_{fluid\ exectuion\ in[0,t)} - \underbrace{allocated(\tau_i, t)}_{real\ execution\ in[0,t)} \tag{4}$$

The generation of an optimal schedule is based on the above definition of *lag*. PFair algorithm does execute all urgent tasks with $lag(\tau_i, t) > 0$ and $lag(\tau_i, t+1) \geq 0$ if τ_i executes. On top of that, PFair algorithm does not execute tnegru tasks, for which $lag(\tau_i, t) < 0$ and $lag(\tau_i, t+1) \leq 0$ if τ_i does not execute. Besides, for other tasks, only those that have the least t such that $lag(\tau_i, t) > 0$ are executed.

The PFair algorithm will assign priorities to tasks at every time slot, which indicated itself as one of job-level dynamic priority scheduling policies. However, this characteristic gives rise to some issues, for example, frequent preemptions and frequent migrations.

Proportionate Fairness algorithm, as a strong candidate for solving resource allocation problems, has wide variety of interesting applications and powerful theoretical support. For instances, Kelly et al [12] presented its application on the problem of rate control for communication networks. Application of PFair algorithm on LANs and hoc networks was fulfilled by Jiang et al [13]. Besides, Bonald's paper [14] provides in-depth queueing analysis over proportionate fairness as well as max-min fairness and balanced fairness.

4.2.2. LLREF

LLREF was firstly introduced by Cho et al [10]. Similar to PFair class of algorithms, LLREF is also based on the fluid scheduling model, where each task executes at a constant rate at all times. The principal idea of LLREF is that given M processors, M largest local remaining execution time tasks are selected first for every secondary event. This is also called the LLREF scheduling policy. To reason over task execution behavior on multiprocessors, a novel abstraction called Time and Local Execution Time Domain Plane (T-L Plane) was developed.

This algorithm divides the schedule into Time and Local execution time planes (TL-planes), which are determined by task deadlines. The algorithm schedules tasks by creating smaller local jobs within each TL-plane. The only parameters considered by the algorithm during a TL-plane are the parameters of the local jobs. When a TL-plane completes, the next TL-plane is started. The duration of each TL-plane is the amount of time between consecutive deadlines.

4.3. Summary

The global scheduling paradigm has advantages over the partitioned approach. First of all, if tasks can join and leave the system at run-time, then it may be necessary to reallocate tasks to processors in the partitioned approach. In addition, the partitioned approach cannot produce optimal real-time schedules – one that meets all task deadlines when task utilization demand does not exceed the total processor capacity – for periodic task sets, since the partitioning problem is analogous to the bin-packing problem which is known to be NP-hard in the strong sense. On top of that, in some embedded processor architectures with no cache and simpler structures, the overhead of migration has a lower impact on the performance. Finally, global scheduling can theoretically contribute to an increased understanding of the properties and behaviors of real-time scheduling algorithms for multiprocessors.

However, the global scheduling paradigm has also several disadvantages. Firstly, global scheduling strategies are much more complicated to implement than partitioned scheduling. In other words, for the partitioned approach, once a set of tasks are allocated to processors, the multiprocessor real-time scheduling problem becomes a collection of single processor real-time scheduling problems. The ease of programming partitioned scheduling is obvious since the single processor scheduling problem has already been well-studied and optimal algorithms with easy implementations already exist. Secondly, migrating tasks at run-time means more runtime overhead in that migrating tasks may suffer cache misses on the newly assigned processor. If the task set is fixed and known in advanced, it is obvious that the partitioned approach provides more appropriate solutions.

5. Hybrid Approaches

6. Conclusions

References

[1] C. L. Liu, J. W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, Journal of the ACM (JACM) 20 (1973) 46–61.

[2] L. Sha, R. Rajkumar, J. P. Lehoczky, Priority inheritance protocols: An approach to real-time synchronization, Computers, IEEE Transactions on 39 (1990) 1175–1185.

[3] A. Mohammadi, S. G. Akl, Scheduling algorithms for real-time systems, School of Computing, Queens University (2005).

[4] R. I. Davis, A. Burns, A survey of hard real-time scheduling for multiprocessor systems, ACM Computing Surveys (CSUR) 43 (2011) 35.

[5] M. R. Garey, D. S. Johnson, Computers and intractability: a guide to the theory of np-completeness. 1979, San Francisco, LA: Freeman (1979).

[6] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, S. Baruah, A categorization of real-time multiprocessor scheduling problems and algorithms, Handbook on scheduling algorithms, methods, and models (2004) 30–1.

[7] S. K. Dhall, C. Liu, On a real-time scheduling problem, Operations research 26 (1978) 127–140.

[8] Y. Oh, S. H. Son, Tight performance bounds of heuristics for a real-time scheduling problem (1993).

[9] A. Khemka, R. Shyamasundar, An optimal multiprocessor real-time scheduling algorithm, Journal of parallel and distributed computing 43 (1997) 37–45.

[10] H. Cho, B. Ravindran, E. D. Jensen, An optimal real-time scheduling algorithm for multiprocessors, in: Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International, IEEE, pp. 101–110.

[11] S. K. Baruah, N. K. Cohen, C. G. Plaxton, D. A. Varvel, Proportionate progress: A notion of fairness in resource allocation, Algorithmica 15 (1996) 600–625.

[12] F. P. Kelly, A. K. Maulloo, D. K. Tan, Rate control for communication networks: shadow prices, proportional fairness and stability, Journal of the Operational Research society (1998) 237–252.

[13] L. B. Jiang, S. C. Liew, Proportional fairness in wireless lans and ad hoc networks, in: Wireless Communications and Networking Conference, 2005 IEEE, volume 3, IEEE, pp. 1551–1556.

[14] T. Bonald, L. Massoulié, A. Proutiere, J. Virtamo, A queueing analysis of max-min fairness, proportional fairness and balanced fairness, Queueing systems 53 (2006) 65–84.