2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum

Fixed-Priority Multiprocessor Scheduling: Critical Instant, Response Time and Utilization Bound

Nan Guan
Uppsala University, Sweden
Email: nan.guan@it.uu.se

Wang Yi
Uppsala University, Sweden
Email: yi@it.uu.se

***Abstract*—The rapid development of multi-core processors leads to a constantly increasing trend of deploying real-time systems on multi-core platforms, to satisfy the dramatically increasing high-performance and low-power requirements. This trend demands effective and efficient multiprocessor real-time scheduling techniques. The uniprocessor scheduling problem has been well studied during the last 40 years. However the multiprocessor scheduling problem to map tasks onto parallel architectures is a much harder challenge. In this work, we study several fundamental problems in multiprocessor scheduling, namely the critical instant, bounded responsiveness, and utilization bound.**

I. Introduction

A real-time system is an information processing system which has to respond to externally generated input stimuli within a finite and specified period: the correctness depends not only on the logical result but also on the time it was delivered; the failure to respond is as bad as the wrong response [12]. These systems are nowadays present in a wide variety of embedded systems and cyber-physical systems, such as automotive/avionic control systems, military applications and environmental monitoring systems. A real-time system typically consists of multiple recurrent processes. These recurrent processes may be invoked with different periods (different frequencies). It is a challenging problem to decide how to arrange the execution of these processes such that all of them can meet their timing requirements at each invocation. This problem is called real-time scheduling.

A milestone of the research on real-time scheduling is the publication of Liu and Layland's seminal paper [31] in 1973, on the topic of real-time scheduling on uniprocessor systems. Significant research efforts and advances have been made upon on Liu and Layland's fundamental framework in the last 40 years. Today, although there is still considerable research going on, uniprocessor real-time scheduling theory can be viewed as reasonably mature, with a large number of key results documented in text books and successfully transferred into industrial practice [17].

The rapid development of multi-core processors leads to a constantly increasing trend of deploying real-time systems on multi-core platforms, to satisfy the dramatically increasing high-performance and low-power requirements. This trend demands effective and efficient techniques for scheduling real-time systems on multi-cores. The problem of scheduling real-time workloads on parallel computing architectures (e.g., multi-core processor) is called real-time multiprocessor scheduling, or multiprocessor scheduling for short. Multiprocessor scheduling theory originates in the late 1960's and early 1970's. [30] noted that multiprocessor real-time scheduling is intrinsically a much more difficult problem than uniprocessor scheduling: "*Few of the results obtained for a single processor generalize directly to the multiple processor case; bringing in additional processors adds a new dimension to the scheduling problem. The simple fact that a task can use only one processor even when several processors are free at the same time adds a surprising amount of difficulty to the scheduling of multiple processors.*"

Ceaseless work has been done on multiprocessor scheduling since 1960's, and a burst of research efforts on this topic start since late 1990's, around the same time as the major silicon vendors such as IBM and AMD start the development of multi-core processors. Nowadays, the multiprocessor platform is gradually becoming the default setting of the real-time scheduling research. Despite the great effort made in the last 40 years, the research on multiprocessor scheduling problem is still far from mature. Many fundamental problems in multiprocessor scheduling are still open.

The thesis of this research is to study the fundamental open problems in multiprocessor scheduling. The target is to as much as possible generalize the classical theoretical results of uniprocessor scheduling to multiprocessor setting, such that the well-established design/analysis framework and tools of uniprocessor real-time systems can be easily adopted and extended to multi-core systems.We focus on a particular subclass of real-time scheduling algorithms, fixed-priority scheduling, in which each process is assigned a static priority and a higher-priority process always has privilege for execution than a lower-priority one. Fixed-priority is the most widely used scheduling paradigm in industrial practise, and is the default scheduler setting in mainstream real-time operating systems.

II. Related Work

Multiprocessor scheduling are usually categorized into two paradigms [14]: global scheduling, in which each task can execute on any available processor at runtime, and partitioned scheduling in which each tasks is assigned to a processor beforehand, and at runtime each task can only execute on this particular processor.

978-0-7695-4676-6/12 $26.00 © 2012 IEEE
DOI 10.1109/IPDPSW.2012.305

Partitioned scheduling enjoys relatively easier design and analysis: as soon as the system has been partitioned into subsystems that will be executed on individual processors each, the traditional uniprocessor real-time scheduling and analysis techniques can be applied to each individual subsystem/ processor. The system partitioning is similar to the bin-packing problem [15], for which many efficient heuristics are known despite of its general intractability. Much work has been done on adopting different bin-packing heuristics to multiprocessor scheduling problem [20], [33], [16], [11]. Similar to the bin-packing problem, partitioning scheduling suffers resource waste due to fragmentation. Theoretically, the worst-case utilization bound of partitioned scheduling can not exceed 50% regardless of the local scheduling algorithm on each processor [14].

On the other hand, global scheduling on average utilizes computing resource better, and is more robust in the presence of timing errors. However, the analysis of global scheduling is significantly more difficult. Global scheduling algorithms based on widely optimal uniprocessor scheduling algorithms like RM and EDF suffer from the so-called Dhall effect [20], namely some system with utilization arbitrarily close to 1 can be infeasible by global RM/EDF scheduling no matter how many processors are added to the system. A major obstacle in precisely analyzing global scheduling and thereby fully exploring its potential is that the critical instant in global scheduling is in general unknown. Several attempts have been made to exhaustively analyze global scheduling by either explicit or symbolic state space enumeration [6], [21], [22], but all run into serious scalability problem. A large body of works have been done in efficient analysis of global scheduling by over-approximation [5], [10], [9], [7], [8].

III. Problem Model

We consider a *sporadic task set* τ consists of N sporadic tasks running on a multiprocessor platform of M processors. We use $\tau_i = \langle C_i,\ D_i, T_i \rangle$ to denote such a task where C_i is the *worst-case execution time* (WCET), D_i is the relative deadline for each release, and T_i is the minimum inter-arrival separation time also referred to as the *period* of the task. The *utilization* of a task τ_i is $U_i = C_i/T_i$. We define the total utilization of the task set as

$$U_{total} = \sum_{\tau_i \in \tau} U_i$$

An *implicit-deadline* task τ_i satisfies the restriction $D_i = T_i$, a *constrained-deadline* task τ_i satisfies $D_i \leq T_i$, whereas an *arbitrary-deadline* task τ_i does not constrain the relation between D_i and T_i. We will consider all types.

We further assume that all tasks are ordered by priorities, i.e., τ_i has higher priority than τ_j iff $i < j$. In this work we consider preemptively scheduling, so a higher priority task always has privilege for execution over a lower priority task.

A sporadic task τ_i generates a potentially infinite sequence of *jobs* with successive job-arrivals separated by at least T_i time units. Each job J adheres to the conditions C_i and D_i of its task τ_i and has additional properties of the *release time*, denoted by r, the *deadline*, denoted by d (derived by $d = r + D_i$, and the *finish time* by f, which is the time instant at which J just finished its execution. We define the *response time* of J as the difference between its release and finish times $R = f - r$. The *worst-case response time* (WCRT) R_i of task τ_i is the maximal response time value among all jobs of τ_i in all job sequences possible in the system.

Since D_i is allowed to be larger than T_i, it is possible that several jobs of a task are *active* (i.e., released but not yet finished) simultaneously. We restrict that a job can execute only if its precedent job has been already finished, to avoid unnecessary working space conflict. This restriction is commonly adopted in the implementation of real-time operating systems for multicores/multiprocessors, for instance, RTEMS [32] and LITMUS^{RT} [13].

IV. Critical Instant

A. Uniprocessor Scheduling

A *critical instant* for a task is defined to be an instant at which a request for that task will have the largest response time [31]. The critical instant of fixed-priority uniprocessor scheduling is presented in Liu and Layland's seminal paper:

Theorem IV.1. *[31] A critical instant for any task occurs whenever the task is requested simultaneously with requests for all higher priority tasks.*

Due to the slight difference between the model in [31] and this work, we shall add two extra (rather trivial) constraints to obtain the critical instant for sporadic tasks:

- Each task always releases jobs as soon as possible.
- Each task always executes for worst-case execution time

The critical instant is one of the most important concept in uniprocessor scheduling analysis. Despite the infinite state-space of a sporadic task set's runtime behavior, the analysis can be limited to one concrete scenario specified by the critical instant. The critical instant of fixed-priority uniprocessor scheduling has also been generalized to different variants of the standard sporadic tasks, e.g., non-preemptive tasks [18], task system shared resource [35], and tasks with offsets [26].

B. Multiprocessor Scheduling

The critical instant of fixed-priority uniprocessor scheduling does not necessarily lead to the worst-case response time in global scheduling. First, the simultaneous release pattern does not necessarily lead to the worst-case response time [34]. Secondly, the as-soon-as-possible release pattern does not necessarily lead to the worst-case response time [4]. Therefore, one can not reduce the analysis to a concrete scenario as in uniprocessor scheduling. One way to conduct the analysis in the presence of an unknown critical instant is to (explicitly or symbolically) enumerate all the possible system behavior [6], [21], [22], which turned out to be very unscalable due to the state-space explosion. In order to analyze real-life scale applications, people resort to efficient approximate analysis, which yields safe but conservative results.

In [23], we developed approximate response time analysis techniques for fixed-priority global scheduling based on the concept of abstract critical instant, which can be describe as follows:

- All higher priority tasks, except $M-1$ of them, is simultaneously requested.
- Each task always releases jobs as soon as possible.
- Each task always executes for worst-case execution time.

The abstract critical instant still does not provide precise information about the worst-case release times of the higher priority tasks, but we are left with a set among which the real critical instant can be found – but this set is significantly smaller than the whole space of possible job sequences.

V. Bounded Response Time

A. *Uniprocessor Scheduling*

The response time of each task is bounded under fixed-priority uniprocessor scheduling for any task system with:

$$U_{total} \leq 1$$

Recall that the utilization $U_i = C_i/T_i$ of a task represents the portion of the processing capacity needed by this task in the long term. So if a task set has total utilization strictly smaller than 1 (or has total utilization equal to 1 but requests less workload than the worst case at runtime), in the long term the total capacity requested by the task set is fewer than what the processor provides, so the busy period (the continuous time interval in which the processor is executing some task) will terminate in a bounded time. If a task set has total utilization equal to 1 and requests exactly the worst-case workload, then under the critical instant (all tasks release together) the total request in the time interval of length equal to the task set's least common period will be finished exactly at the end of this interval, which implies the response time of each task is also bounded.

B. *Multiprocessor Scheduling*

In multiprocessor scheduling, the condition

$$U_{total} \leq M$$

implies the total capacity does not exceed what provided by the processor platform. However, such a condition can not guarantee the bounded responsiveness under fixed-priority global scheduling (neither the weaker condition $U_{total} < M$) [19]. Therefore, it leaves open that under what kind of condition the response time of a task is guaranteed to be bounded by fixed-priority global scheduling.

In [23], we addressed the bounded responsiveness problem of fixed-priority global scheduling. More specifically, we first developed the general response time analysis techniques for arbitrary deadline sporadic task systems, and then establish the condition for the termination of such an analysis procedure. The results can be summarized as follows:

Theorem V.1. *The response time of a task τ_i is bounded if τ_i satisfied the following condition:*

$$\sum_{i<k} V_i^k + M \times U_k < M$$

where $V_i^k = \min(U_i, 1-U_k)$ is the interfering utilization*.*

Intuitively, the item $M \times U_k$ addresses the resource waste in the situation that all the other $M-1$ processors are idle when the analyzed task is executing. The interfering utilization V_k^i restricts the utilization which is relevant for interference to the part that is not running in parallel to the task in question.

VI. Utilization Bound

A. *Uniprocessor Scheduling*

Utilization bound of a scheduling algorithm is a value UB such that any task set whose total utilization U_{total} not exceeding UB is guaranteed to be schedulable. In the seminal paper [31], Liu and Layland discovered the utilization bound for (the optimal algorithm of) fixed-priority uniprocessor scheduling with implicit-deadline sporadic task systems:

Theorem VI.1. *[31] A task set is schedulable by Rate Monotonic (RM) scheduling if it holds*

$$U_{total} \leq N \times (2^{\frac{1}{N}} - 1)$$

The term $N \times (2^{\frac{1}{N}} - 1)$ is known as the famous Liu and Layland utilization bound, which is monotonically decreasing with respect to N and approaches 0.693 as N goes to infinity. RM scheduling is the optimal fixed-priority uniprocessor scheduling algorithm, and Liu and Layland utilization bound is tight. So $N \times (2^{\frac{1}{N}} - 1)$ is the tight utilization bound for fixed-priority uniprocessor scheduling in general.

B. *Multiprocessor Scheduling*

The utilization bound concept can be extended to multiprocessor scheduling: the utilization bound UB guarantees that any task set whose total utilization normalized by the number of processors ($\frac{U_{total}}{M}$) is bounded by UB is schedulable. Multiprocessor scheduling can be categorized into global scheduling and partitioned scheduling. The standard global scheduling version of RM suffers the so-called Dhall's effect [20], namely some system with utilization arbitrarily close to 1 can be unschedulable by global RM scheduling no matter how many processors are added to the system, which implies the utilization bound of global RM is arbitrarily close to zero. The best known utilization bound of fixed-priority global scheduling is 0.38 [2]. On the other hand, the utilization bound of partitioned scheduling cannot exceed 0.5, which is the same limit as in bin-packing problem. So neither global nor partitioned scheduling achieves as high utilization bound as in uniprocessor scheduling.

In [24], we developed a fixed-priority multiprocessor scheduling algorithm SPA that generalizes the Liu and Layland utilization bound to multiprocessor:

Theorem VI.2. *A task set is schedulable by SPA on M processors if it holds*

$$\frac{U_{total}}{M} \leq N \times (2^{\frac{1}{N}} - 1)$$

SPA is in the category of semi-partitioned scheduling or partitioned scheduling with task splitting [1], [27], [3], [28], [29], in which most tasks are statically assigned to one fixed processor as in partitioned scheduling, while a few number of tasks are split into several subtasks, which are assigned to different processors. Bin-packing problem will become trivial if item splitting is allowed. However, semi-partitioned scheduling is a much more difficult problem. The challenge is that the scheduling algorithm must guarantee the serial execution of a split task on different processors.

SPA uses the worst-fit bin-packing heuristics and increasing priority order to allocate (a portion of) each task to processors, and uses RM as the local scheduling on each processor. The main insight for SPA to achieve the same (optimal) utilization bound as in uniprocessor scheduling is that, the worst-fit bin-packing and increasing priority order guarantees that task splitting happens only with relatively higher priority tasks. The higher priority tasks have larger slack, and thereby can tolerant extra timing constraints for guaranteeing the serial execution of different parts of a split task on different processors. This result has been extended to parametric utilization bounds that guarantees better resource usage by exploring the knowledge of task parameters [25].

VII. Future Work

The following topics are in the scope our potential further work directions:

- To further refine the abstract critical instant, in order to obtain more precise response time analysis.
- To extend the so far obtained utilization bound results to heterogenous multiprocessor platforms. Heterogenous multiprocessor can be classified into uniformly heterogenous and unrelated heterogenous multiprocessor platforms [14]. Our preliminary work indicates the extension of our results to unrelated heterogenous multiprocessors would be especially challenging.
- To study the analysis of multiprocessor scheduling with more general task models. Many applications deployed on multi-cores are parallel programs, which can be represented by task graph models. We shall study extend our abstract critical instant and response time analysis techniques to these parallel applications.

References

[1] J. Anderson, V. Bud, and U. Devi. An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In *ECRTS*, 2005.

[2] B. Andersson. Global static priority preemptive multiprocessor scheduling with utilization bound 38%. In *OPODIS*, 2008.

[3] B. Andersson, K. Bletsas, and S. Baruah. Scheduling arbitrary-deadline sporadic task systems multiprocessors. In *RTSS*, 2008.

[4] B. Andersson and J. Jonsson. Some insights on fixed-priority preemptive non-partitioned multiprocessor scheduling. *Technical Report, Chalmers University of Technology.*, 2001.

[5] T. P. Baker. Multiprocessor EDF and deadline monotonic schedulability analysis. In *RTSS*, 2003.

[6] T. P. Baker and M. Cirinei. Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. In *OPODIS*, 2007.

[7] S. K. Baruah. Techniques for multiprocessor global schedulability analysis. In *RTSS*, 2007.

[8] S. K. Baruah and T. P. Baker. Schedulability analysis of global EDF. In *Real Time Systems*, 2008.

[9] M. Bertogna and M. Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *RTSS*, 2007.

[10] M. Bertogna, M. Cirinei, and G. Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *ECRTS*, 2005.

[11] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son. New strategies for assigning real-time tasks to multiprocessor systems. In *IEEE Transactions on Computers*, 1995.

[12] A. Burns and A. Wellings. Real-time systems and programming languages. In *Addison-Wesley, 3rd edition*, 2001.

[13] J. Calandrino, H. Leontyev, A. Block, U. Devi, and J. Anderson. Litmusrt: A testbed for empirically comparing real-time multiprocessor schedulers. In *RTSS*, 2006.

[14] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. *A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms*. 2004.

[15] E. G. Coffman, M. R. Garey, and D. S. Johnson. *Approximation algorithms for bin packing: a survey*. 1997.

[16] S. Davari and S. K. Dhall. On a periodic real time task allocation problem. In *Annual International Conference on System Sciences*, 1986.

[17] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. In *ACM Computer Survey*, 2011.

[18] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. In *Real-Time Systems*, 2007.

[19] U. Devi and J. Anderson. Tardiness bounds for global EDF scheduling on a multiprocessor. In *RTSS*, 2005.

[20] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. In *Operations Research, Vol. 26, No. 1, Scheduling*, 1978.

[21] N. Guan, Z. Gu, Q. Deng, S. Gao, and G. Yu. Exact schedulability analysis for static-priority global multiprocessor scheduling using model-checking. In *SEUS*, 2007.

[22] N. Guan, Z. Gu, M. Lv, Q. Deng, and G. Yu. Exact schedulability analysis of global scheduling on multiprocessor platforms by symbolic model checking. In *ISORC*, 2008.

[23] N. Guan, M. Stigge, W. Yi, and G. Yu. New response time bounds of fixed priority multiprocessor scheduling. In *RTSS*, 2009.

[24] N. Guan, M. Stigge, W. Yi, and G. Yu. Fixed-priority multiprocessor scheduling with Liu & Layland's utilization bound. In *RTAS*, 2010.

[25] N. Guan, M. Stigge, W. Yi, and G. Yu. Parametric Utilization Bounds for Fixed-Priority Multiprocessor Scheduling. In *IPDPS*, 2012.

[26] M. G. H. J. Palencia Gutierrez. Schedulability analysis for tasks with static and dynamic offsets. In *RTSS*, 1998.

[27] S. Kato and N. Yamasaki. Real-time scheduling with task splitting on multiprocessors. In *RTCSA*, 2007.

[28] S. Kato and N. Yamasaki. Portioned EDF-based scheduling on multiprocessors. In *EMSOFT*, 2008.

[29] K. Lakshmanan, R. Rajkumar, and J. Lehoczky. Partitioned fixed-priority preemptive scheduling for multi-core processors. In *ECRTS*, 2009.

[30] C. L. Liu. Scheduling algorithms for multiprocessors in a hard real-time environment. In *JPL Space Programs Summary*, 1969.

[31] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. In *Journal of the ACM*, 1973.

[32] O.-L. A. R. C. (OAR). RTEMS Applications C User's Guide. 2001.

[33] Y. Oh and S. H. Son. Allocating fixed priority periodic tasks on multiprocessor systems. In *Real-Time Systems*, 1995.

[34] R. M. S. Lauzac and D. Mosse. Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor. In *ECRTS*, 1998.

[35] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. In *IEEE Transactions on Computers*, 1990.