Generalized Feasibility Analysis on Uniform Multiprocessors

Shelby Funk
The University of Georgia
shelby@cs.uga.edu

Abstract

This paper presents feasibility tests for scheduling instances of real-time jobs on uniform multiprocessor platforms. In a uniform multiprocessor, each processor has an associated *speed* – a job executing on a processor of speed s for t time units performs $s \times t$ units of work. Two feasibility tests are presented. First, a sufficient condition is presented – whenever this condition holds the jobs are known to be feasible on the multiprocessor. Then, a necessary condition is presented – whenever the jobs are feasible on the multiprocessor, this condition must hold.

1 Introduction

In real-time systems, jobs are submitted with specific timing requirements. These requirements are often expressed as deadlines. A job with relative deadline D must complete all its execution requirement no more than D time units after it arrives. Given a set of jobs and a specific processing platform, *feasibility analysis* is the practice of determining whether it is possible to schedule *all* the jobs in such a way that they all meet their deadlines.

This paper introduces feasibility tests for hard real-time instances on uniform multiprocessors. In uniform multiprocessors, each processor has an associated speed – if a processor of speed s executes for t time units, it performs $s \times t$ units of work. There are two tests presented below – a *necessary* test and a *sufficient* test. The necessary test presents conditions that must hold for *any* feasible real-time system executing independent jobs on a uniform multiprocessor. The sufficient test presents a condition that, when satisfied, ensures that *some* schedule exists that can execute all jobs without missing any deadlines.

One aspect of a scheduling algorithm that a designer must take into consideration is whether or not to allow *preemption* – i.e., can a currently executing job be interrupted and start again later. While there is overhead incurred by allowing preemption, systems that do not allow preemption may require extra slack. This is because a job with a very long execution requirement may begin executing just before the arrival of a job with a very short deadline. The analysis in this paper assumes that preemption is permitted. Another aspect that a designer must consider when scheduling on multiprocessors is whether or not to allow *migration* – i.e., if a job is preempted, can it resume execution on a different processor. The analysis in this paper assumes that migration is not allowed.

2 Model and definitions

Job model. A *real-time job* J_i is described by the three-tuple $(J_i.A, J_i.E, J_i.D)$, where $J_i.A$ is the job's arrival time, $J_i.E$ is its worst case execution requirement, and $J_i.D$ is it's relative deadline – job J_i must be allowed to execute $J_i.E$ units of execution within the time interval $[J_i.A, J_i.A + J_i.D]$. A *real-time instance* $I = \{J_1, J_2, ..., J_n, ...\}$ is a collection of real-time jobs. Without loss of generality, we assume jobs are indexed in weakly increasing order according to relative deadline – i.e., whenever $i < j$, we have $J_i.D \leq J_j.D$. Note that this is a very general model of computation – these jobs may be generated by periodic [4], sporadic [3], multiframe [5, 6], or generalized multiframe [2] tasks. The only assumption we make is that the jobs are *independent* – i.e., the jobs may execute in any order. Also, we assume the jobs may be preempted without incurring extra cost (or, to be more accurate, we assume the costs associated with preemption are included in the execution requirements).

Tasks are analyzed using three parameters: density, demand, and load. The density is the maximum execution requirement per time unit of any job. More

precisely,

$$\mathsf{density}(I) = \max_{J_i \in I} \frac{J_i.E}{J_i.D}. \quad (1)$$

Given any time interval $[t_s, t_f]$, the demand over that interval is the minimum amount of work that *must* be done during that time interval. We find the minimum amount of work by considering only those jobs that arrive and have their deadline in the interval $[t_s, t_f]$.

$$\mathsf{demand}(I, t_s, t_f) = \sum_{[J_i.A, J_i.A+J_i.D] \subseteq [t_s, t_f]} J_i.E. \quad (2)$$

Finally, I's load is the maximum average demand over all intervals.

$$\mathsf{load}(I) = \max_{t_s < t_f} \frac{\mathsf{demand}(I, t_s, t_f)}{t_f - t_s}. \quad (3)$$

Multiprocessor model. For the analysis that follows, the instance I executes on a *uniform multiprocessor platform*, in which the processors execute at different speeds than one another. The m-processor multiprocessor π is described by an ordered list of speeds $\pi = [s_1, s_2, ..., s_m]$. Each speed s_i indicates the amount of work the processor performs in one unit of time. Hence, if a job J_i executes on processor[1] s_j, the job will complete execution after $J_i.e/s_j$ units of time. Without loss of generality, we assume the processor speeds are listed in weekly decreasing order – i.e., whenever $i < j$, we have $s_i \geq s_j$. The total processing speed of π is denoted $S(\pi)$ – i.e., $S(\pi) = \sum_{i=1}^{m} s_i$. This is the maximum amount of work that can be performed on π in one unit of time.

Job to processor assignment algorithm. Recall we are looking for feasibility tests assuming jobs do not migrate among processors. We perform the analysis below assuming jobs are assigned to processors using the algorithm JobAssign introduced by Baruah and Fisher [1]. This algorithm assigns jobs to processors prior to execution. Job assignments are done in order of job index. Hence, jobs with shorter relative deadlines are assigned to processors before jobs with longer relative deadlines. If at any point, the algorithm attempts to assign a job to a processor and no processors can schedule the job without missing some deadline, then the algorithm JobAssign fails. Note that this algorithm only determines *which processor* the jobs executes on. It does not determine *when* the job executes. Each processor will determine when jobs execute using a local scheduling algorithm such as the Earliest Deadline First (EDF) algorithm, in which jobs with earlier deadlines are given higher priority than those with later deadlines.

[1]In a mild abuse of notation, we let s_j denote both the processor and the speed at which the processor executes.

3 Feasibility Conditions

This section introduces two feasibility tests. Any system satisfying the first test is guaranteed to be feasible. Furthermore, jobs can be feasibly assigned to processors using the JobAssign task assignment. By contrast, any feasible system must satisfy the second test. However, satisfying this second test does not guarantee feasibility. Thus, this test will most likely be used to determine if a system is *in*feasible.

Baruah and Fisher [1] developed tests similar to the ones presented in this paper. However, they considered an *identical* multiprocessor model, in which all processors execute at the same speed. In their work, they assume that all execution requirements are normalized to the processing speed – hence the processor speeds are all assumed to be 1. We will see that Baruah and Fisher's results are corollaries to the results presented in this paper.

3.1 A Sufficient Test

In this section, we introduce a condition that ensures a system can be feasibly scheduled.

Theorem 1 *Let I be any real-time instance and $\pi = [s_1, s_2, ..., s_m]$ be any m-processor uniform multiprocessor. If the following condition is satisfied*

$$\mathsf{load}(I) \leq \frac{1}{3}\Big(S(\pi) - (m-1)\mathsf{density}(I)\Big). \quad (4)$$

then I can be feasibly scheduled on π. Furthermore, jobs can be feasibly assigned to processors using algorithm JobAssign.

Proof: The proof is by contradiction. We assume that I cannot be feasibly scheduled on π and show that Condition 4 does not hold. Assume that all the jobs in $I = \{J_1, J_2, ..., J_x\}$ except J_x can be feasibly assigned to some processor in π. Let T_x be the interval $[J_x.A, J_x.A + J_x.D]$. For $1 = 1, 2, ..., m$, let I_i be the jobs of I that are assigned to processor s_i and let W_i be the minimum amount of work that must be executed within the interval T_x without causing I_i to become infeasible. Since J_x cannot be assigned

to any processor, the following must hold for each processor

$$W_i + J_x.E > J_x.D \cdot s_i$$

Therefore,

$$\sum_{i=1}^{m} W_i > S(\pi) \cdot J_x.D - m \cdot J_x.E. \quad (5)$$

Let I_W be the set of jobs in I that contribute to the value W_i for some i. Let J_s and J_f be jobs in I_W with the earliest arrival time and latest deadline, respectively. Consider the interval $[J_s.A, J_f.A + J_f.D]$. By definition of load,

$$\mathsf{load}(I) \geq \frac{\mathsf{demand}(I, J_s.A, J_f.A + J_f.D)}{J_f.A + J_f.D - J_s.A}.$$

$J_x.E$ is not included in W_i for any i. Therefore, $\mathsf{demand}(I, J_s.A, J_f.A + J_f.D) - J_x.E$ provides an upper bound for the left hand side of Inequality 5. Combining this with the load bound above, we have

$$(J_f.A + J_f.D - J_s.A)\mathsf{load}(I) - J_x.E > S(\pi) \cdot J_x.D - m \cdot J_x.E. \quad (6)$$

We can divide the interval $[J_s.A, J_f.A+J_f.D]$ into three subintervals defined by the arrival and deadline of J_x — the portion before J_x's arrival, $T_{s \text{ to } x} = [J_s.A, J_x.A]$, the portion between J_x's arrival time and deadline, T_x, and the portion after J_x's deadline, $T_{x \text{ to } f} = [J_x.A + J_x.D, J_f.A + J_f.D]$. Since J_s must execute within T_x, its deadline must be after the arrival of J_x – i.e., $J_s.A + J_s.D > J_x.A$. Also, since jobs are indexed according to relative deadline $J_s.D \leq J_x.D$. Therefore, the length of $T_{s \text{ to } x}$ is less than $J_x.D$. Similarly, the length of $T_{x \text{ to } f}$ is less than $J_x.D$. Therefore, the length of the entire interval is less than $3J_x.D$,

$$J_f.A + J_f.D - J_s.A < 3 \cdot J_x.D$$

Combining this with Inequality 6 gives

$$3J_x.D \cdot \mathsf{load}(I) - J_x.E > S(\pi) \cdot J_x.D - m \cdot J_x.E,$$

which implies

$$\mathsf{load}(I) > \frac{1}{3}\left(S(\pi) - (m-1)\frac{J_x.E}{J_x.D}\right).$$

Recall the density is the maximum ratio of execution requirement to relative deadline. Therefore,

$$\mathsf{load}(I) > \frac{1}{3}\Big(S(\pi) - (m-1)\mathsf{density}(I)\Big).$$

Hence, whenever I is infeasible on π, Inequality 4 is violated. ■

Corollary 1 (Baruah and Fisher, 2006 [1]) *Let I denote a real-time instance satisfying*

$$\mathsf{load}(I) \leq \frac{1}{3}\Big(m - (m-1)\mathsf{density}(I)\Big).$$

Algorithm JobAssign *successfully assigns every job to some processor on a platform comprised of m unit-speed processors.*

Proof: This proof follows from Theorem 1. Since there are m unit-speed processors, $S(\pi) = m$. Therefore, Inequality 4 becomes $\mathsf{load}(I) \leq \frac{1}{3}\Big(m - (m-1)\mathsf{density}(I)\Big)$. ■

3.2 A Pair of Necessary Conditions

The following theorem provides a pair of conditions that *must* hold for any real-time instance to be feasible on a uniform multiprocessor π.

Theorem 2 *Let I be any real-time instance and $\pi = [s_1, s_2, ..., s_m]$ be any uniform multiprocessor. If I is feasible on π, the following two conditions must hold.*

$$\mathsf{density}(I) \leq s_1, \text{and} \quad (7)$$

$$\mathsf{load}(I) \leq S(\pi). \quad (8)$$

Proof: We assume I is feasibly scheduled on π and show both the above conditions must hold.

Let J_m be any job in I and assume the algorithm assigns J_m to processor s_i. Let T be the total time J_m executes on s_i. Since J_m is able to meet its deadline, T must be at most $J_m.D$. Therefore, $J_m.E \leq J_m.D \times s_i$. Dividing by $J_m.D$ gives $J_m.E/J_m.D \leq s_i$. Since this inequality must hold for *every* job in I, we must have $\max_{J_m \in I}\{J_m.E/J_m.D\} \leq \max_{i \leq m} s_i$ – i.e., $\mathsf{density}(I) \leq s_1$.

Now assume I is feasible on π and let $[t_s, t_f]$ be an interval with maximum average demand – namely, $\mathsf{load}(I) = \mathsf{demand}(I, t_s, t_f)/(t_f - t_s)$. By the definition of demand (Equation 2),

$$\mathsf{load}(I) = \frac{\sum_{[J_i.A, J_i.A+J_i.D] \subseteq [t_s, t_f]} J_i.E}{t_f - t_s}.$$

Because $S(\pi)$ is the maximum amount of work that π can complete in one time unit, the total amount of work that π can complete in the interval $[t_s, t_f]$ is $S(\pi) \times (t_f - t_s)$. Since I is feasible on π the demand in this interval is met. Therefore,

$$\sum_{[J_i.A, J_i.A+J_i.D] \subseteq [t_s, t_f]} J_i.E \leq S(\pi) \times (t_f - t_s).$$

Dividing both sides by $(t_f - t_s)$ gives $\mathsf{load}(I) \leq S(\pi)$. ■

Corollary 2 (Baruah and Fisher, 2006 [1]) *If a real-time instance I is feasible on an identical multiprocessor platform comprised of m unit-speed processors, it must be the case that*

$$\text{(i) } \mathsf{density}(I) \leq 1 \text{ and (ii) } \mathsf{load}(I) \leq m.$$

Proof: Since all processors in the identical multiprocessor have a speed of 1, we see that $s_1 = 1$ and $S(\pi) = m$ on this system. ■

While any feasible system must satisfy the conditions in Theorem 2, there are systems that satisfy these conditions but are not feasible. Consider the following example.

Example 1 Let $I = \{J_1 = J_2 = (0, 1, 1)\}$. Then $\mathsf{density}(I) = 1$ and $\mathsf{load}(I) = 2$. Consider the multiprocessor $\pi = [1, 0.5, 0.5]$. Clearly both conditions in Theorem 2 are satisfied, but I cannot be feasibly scheduled on π since at least one of the jobs would have to simultaneously execute on both of the speed-0.5 processors and job parallelism is not permitted. ■

4 Conclusions and Future Work

This paper has introduces two tests for scheduling real-time instances on uniform multiprocessors. The first test is a sufficient condition that guarantees a given system is feasible. The second test is a necessary pair of conditions that must hold for any feasible system. These tests can be applied to a wide variety of systems provided the jobs are independent. In the future, we would like to tighten these tests and analyze how tight they are. It is fairly easy to tighten the necessary conditions if the instance I is generated by a periodic [4] or sporadic [3] task set. In these systems, the density of I would be the maximum utilization of the task set. We can extend the requirement that $\mathsf{density}(I) \leq s_1$ to a utilization test applying to multiple processors as follows:

$$\sum_{j=1}^{i} u_j \leq s_i \text{ for all } i \leq m$$

where u_j is the utilization of the j^{th} task (in this case, we assume the tasks are indexed in weakly decreasing order of utilization). This result does not directly extend to more general real-time instance because, unlike the case with recurring tasks, the job with the second-highest density may have no impact on the job with the highest density. In our future work, we would like to explore how to generalize the above test to more types of real-time instances.

In Baruah and Fisher's work, the authors explored how tight their feasibility tests are. They concluded that the sufficient feasibility test is satisfied for at least 25% of the feasible task sets (more precisely, the $(\mathsf{density}, \mathsf{load})$-space satisfying Corollary 1 is at least 25% of the space satisfying Corollary 2). In future work, we would like to perform similar analysis on the tests presented in this paper. It is fairly easy to see that this ratio is improved by considering a uniform multiprocessor model as opposed to an identical multiprocessor model. We plan to explore how much better we can do when we allow speeds to differ from one another.

References

[1] Baruah, S., and Fisher, N. The feasibility analysis of multiprocessor real-time systems. *EuroMicro Conference on Real-Time Systems (ECRTS)* (2006), 85–96.

[2] Baruah, S. K., Chen, D., Gorinsky, S., and Mok, A. K. Generalized multiframe tasks. *Real-Time Systems: The International Journal of Time-Critical Computing 17*, 1 (July 1999), 5–22.

[3] Dertouzos, M., and Mok, A. K. Multiprocessor scheduling in a hard real-time environment. *IEEE Transactions on Software Engineering 15*, 12 (1989), 1497–1506.

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

[5] Mok, A. K., and Chen, D. A multiframe model for real-time tasks. In *Proceedings of the 17th Real-Time Systems Symposium* (Washington, DC, 1996), IEEE Computer Society Press.

[6] Mok, A. K., and Chen, D. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering 23*, 10 (October 1997), 635–645.