technische universität dortmund
fakultät für informatik
computer science 12

Prof. Jian-Jia Chen

Real-Time Systems (SS 2014)

Exercise 6: Multiprocessor Partitioned Scheduling

Discussion Date: 08, July, 2014

Exercise 6.1

(a) Explain the Makespan problem and the Bin Packing problem. What are their connections to the multiprocessor partitioned scheduling problem for sporadic real-time tasks with implicit deadlines?

(b) Mr. Smart wants to define the monotonicity of the largest utilization first (LUF) algorithm. He has two options in his mind:

- If LUF derives a feasible task partition (with respect to schedulability under EDF) for an input task set, LUF also gives a feasible task partition if the execution time of certain task is reduced.
- If LUF derives a feasible task partition for an input task set, the resulting task partition remains feasible (with respect to schedulabilty under EDF) if the execution time of certain task is reduced.

Help him clarify the correctness of these statements.

Exercise 6.2

Implicit-Deadline Task Partitioning: Task partitioning on M identical multiprocessor systems for rate-monotonic scheduling (implicit-deadline sporadic tasks) can be done by using the hyperbolic bound $\Pi_{\tau_i \in \mathcal{T}_m}(U_i + 1) \leq 2$, where $\mathcal{T}_m$ is the set of tasks assigned on processor m.

- Describe a task partitioning algorithm by extending the LUF algorithm in the lecture and using the hyperbolic bound to decide whether a task can be assigned on a processor m. (*Hint: you don't need to sort the tasks based on their utilizations*)

- Explain the condition
$$(U_k + 1)^M \Pi_{i<k}(U_i + 1) > 2^M$$
of failure of the above algorithm when assigning task τ_k on M processors, in which the task assignment starts from a predefined order $\tau_1, \tau_2, \ldots, \tau_k$.

- Explain that the speed-up factor of the above algorithm is 2.668 when k is sufficiently large. (*Hint: the solution x for the following equation $(1 + 1/x) \cdot e^{1/x} - 2 = 0$ is about 2.668, where e is Euler's number.*)

Exercise 6.3

Constrained-Deadline Task Partitioning: Extend the deadline-monotonic partitioning for fixed-priority scheduling when considering sporadic tasks with constrained deadlines. What is the resource augmentation (speed-up) factor?

Exercise 6.4

Gap between Using More Processors and Speeding-Up:

- What is the difference between using more processors from speeding-up?

- Mr. Smart claims the following statement when considering task partitioning in identical multiprocessor systems: "If there exists a polynomial-time algorithm A which guarantees an augmentation factor for using at most ρ times processors, this algorithm can be converted to a polynomial-time algorithm with a speed-up factor $\lceil \rho \rceil$." Is he correct? How about arguing in the reverse manner?

- Consider deadline-monotonic partitioning for EDF scheduling. When considering speeding up, we have shown in the lecture that the speed-up factor is 4, no matter which fitting algorithm (best-fit, worst-fit, and first-fit) is used. Let's use the same algorithm with the following *best-fit* strategy: choose the processor that has the maximum (total) approximate demand bound function at time D_i when assigning task τ_i if the (total) approximate demand bound function at time D_i is feasible. If none of the current processors can accomodate this task, a new processor is allocated and task τ_i is assigned to this processor.

  Suppose that K is divisible by 2 and the number of tasks is $2K$. We consider an input instance as follows:

  - Let $D_1 = 1$, $C_1 = \epsilon$, and $T_1 = K^K$, with $1 > \epsilon > 0$.
  - For any $i = 2, 4, 6, \ldots, 2K$, let $D_i = K^{\frac{i}{2}-1}$, $C_i = K^{\frac{i}{2}-2}$, and $T_i = D_i$.
  - For any $i = 3, 5, 7, \ldots, 2K-1$, let $D_i = K^{\frac{i-1}{2}}$, $C_i = K^{\frac{i-1}{2}} - K^{\frac{i-1}{2}-1}$, and $T_i = K^K$.

  How many processors do we use by adopting the above deadline-monotonic partitioning and approximate demand bound function for EDF scheduling? Is it feasible to use only two processors by allocating τ_i to processor $(i \mod 2) + 1$? What can we claim about the resource augmentation factor (with respect to using more processors) of the above algorithm?