

CSCE 990: Real-Time Systems

Multiprocessor and Distributed Systems

Steve Goddard
goddard@cse.unl.edu

<http://www.cse.unl.edu/~goddard/Courses/RealTimeSystems>

Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 1

Multiprocessor/Distributed Systems

- ◆ Many real-time systems contain more than a single processor.
- ◆ A multiprocessor system is tightly coupled and usually has shared-memory.
 - » There may be one scheduler for all processors, but this type of scheduling is usually NP-Hard.
 - » Or, there may be a scheduler for each processor, which is the model we will assume.
- ◆ A distributed system is loosely coupled; it is costly to keep global status. There is usually one scheduler for each processor.

Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 2

Multiprocessor/Distributed Systems

- ◆ We assume each processor has its own scheduler and local resources.
- ◆ Local resources may be shared globally, but access to them is controlled locally.
- ◆ All of the same problems with scheduling and shared resources that we discussed on single processors exist on multiprocessor and distributed systems, but the problems are even harder to solve!

Steve Goddard

Real-Time Systems

Multiprocessor & Dis. Systems - 3

Multiprocessors & Resource Access Control

(Chapter 9 of Liu)

- ◆ We first consider how to adapt the analysis discussed previously when tasks access **globally shared resources** from multiple processors in a tightly-coupled systems.
- ◆ Later, in our discussion of distributed systems, we will consider tasks that have **precedence constraints**, and the task may execute in a distributed system.

Steve Goddard

Real-Time Systems

Multiprocessor & Dis. Systems - 4

A Review of Shared Resources

- ◆ In multiprocessor systems we will assume a resource is “hosted” on one processor, but accessible from each processor.
- ◆ We add to the model a set of p serially **reusable** **resources** R_1, R_2, \dots, R_p , where there are v_i units of resource R_i .
 - » Examples of resources:
 - Binary semaphore, for which there is one unit.
 - Counting semaphore, for which there may be many units.
 - Reader/writer locks.
 - Printer.
 - Remote server.

Steve Goddard

Real-Time Systems

Multiprocessor & Dtm. Systems - 5

A Review of Conflicts

- ◆ Two jobs have a **resource conflict** if some of the resources they require are the same.
 - » Note that if we had reader/writer locks, then notion of a “conflict” would be a little more complicated.
- ◆ Two jobs **contend** for a resource when one job requests a resource that the other job already has.
- ◆ The scheduler will always deny a lock request if there are not enough free units of the resource to satisfy the request.

Steve Goddard

Real-Time Systems

Multiprocessor & Dtm. Systems - 6

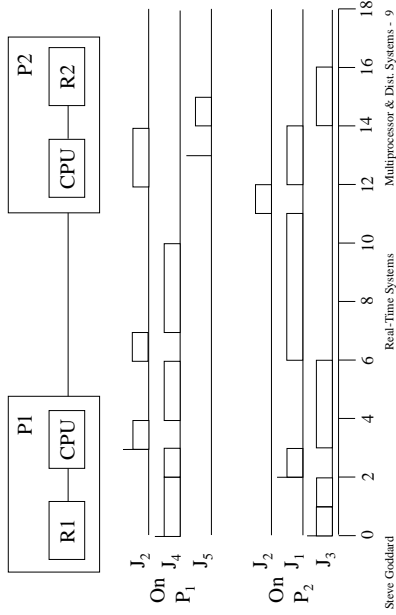
MPCP

- ◆ The Multiprocessor Priority-Ceiling Protocol (MPCP) was created by Sha, Rajkumar, and Lehoczky (1988).
- ◆ It assumes that tasks and resources are statically bound to processors.
 - » The host processor for a resource is called the synchronization processor for that resource.
- ◆ The fixed priority scheduler for each synchronization processor knows the priorities and resources requirements of all tasks requiring access to its globally shared resources.

MPCP (continued)

- ◆ Access to globally shared resources is controlled locally on the synchronization processor according to the Priority-Ceiling Protocol (PCP) we covered earlier, with the following exceptions:
 - » Access to a globally shared resource is modeled as the task executing a global critical section on the synchronization processor for the resource.
 - » All global critical sections are executed at higher priorities than local tasks on the synchronization processor.

Simple MPCP Example



Blocking Time

- ◆ In MPCP, there are now five types of blocking that task T_i may incur:
 - » local blocking time (LBT) due to contention for resources on its local processor;
 - » local preemption delay (LPD) due to the preemption of T_i by global critical sections that belong to remote tasks but execute on its local processor;
 - » remote blocking time (RBT) due to contention with some lower-priority tasks for remote resource(s) on the synchronization processor(s) of the resource(s);
 - » remote preemption delay (RPD) due to preemptions by higher-priority global critical section on synchronization processors of the remote resources required by T_i ; and
 - » deferred blocking time (DBT) due to the suspended execution of local higher-priority tasks.

Bounding Blocking Time

- ◆ In general, the total blocking time $b_i(rc)$ incurred by a job of task T_i is the sum of the previous five blocking terms:

$$b_i(rc) = LBT_i + LPD_i + RBT_i + RPD_i + DBT_i$$

- ◆ Before presenting upper bounds for each of these blocking terms, we need to define a bunch of notation...

» Note: in the notation that follows we have dropped the \wedge symbol above the letter c for various execution cost parameters.

Notation

k_i :	the number of remote critical sections of T_i
P_L :	the local processor of T_i
P_j :	$1 \leq j \leq k_i$, the synchronization processor of the j^{th} remote critical section of each job in T_i
$T(P_L, gcs)$:	subset of all tasks that have global cs on P_L
$T(P_L, gcs, remote)$:	subset containing all the remote tasks in $T(P_L, gcs)$
$T(P_L, gcs, lower)$:	subset containing the lower priority local tasks in $T(P_L, gcs)$
$T(P_j, gcs, lower)$:	the subset of all tasks that have global cs on P_j and have lower priorities than T_i
$T(gcs, higher)$:	subset of equal or higher priority remote tasks that have global cs on any remote synchronization processor P_j for $1 \leq j \leq k_i$, on which T_i executes
$T(\text{local, higher})$:	subset containing all higher priority local tasks

Notation (continued)

- c_L : the blocking time of T_i caused by lower priority local tasks on the local processor P_L
- $c_{k,local}(P_L)$: the total execution time of all global cs that belong to another task T_k and execute on processor P_L
- $c_{k,max}(P_L)$: the maximum execution time of all global cs that belong to T_k and execute on processor P_L
- $c_{k,max}(P_j)$: the maximum execution time of all global cs of T_k on processor P_j
- $c_{k,local}^i$: the total execution time of all global cs of T_k that execute on any P_j , for $1 \leq j \leq \kappa_r$
- $e_{k,L}$: the maximum execution time of the portion of each job in local task T_k that is executed on the local processor P_L
- suspension_time_of_ T_k : the total maximum amount of time for which any job in a local task T_k may be suspended on P_L while it waits for its remote global critical sections to complete.

Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 13

Local Blocking Time (LBT)

- ◆ Recall that according to PCP, each job in T_i may be blocked once by a lower-priority local task.
- ◆ When a job uses a remote resource, it gives up the local processor and may be blocked again by some lower-priority local task (equivalent to self-suspension).
- ◆ Thus, for task T_i :
$$LBT \leq (\kappa_r + 1)c_L$$
- ◆ If P_i is the only processor in the system, $\kappa_r = 0$, and the blocking time of T_i due to resource contention is at most c_L , which can be found as before.

Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 14

Local Preemption Delay (LPD)

- ◆ LPD accounts for the amount of time task T_i is preempted by global critical sections of tasks in $\mathbf{T}(P_L, \text{gcs, remote})$ and $\mathbf{T}(P_L, \text{gcs, lower})$:
 - » During each period of T_i , each global cs of a task T_k in $\mathbf{T}(P_L, \text{gcs, remote})$ may execute at most $\left\lfloor \frac{p_i}{p_k} \right\rfloor + 1$ times.
 - » A job of T_i can be delayed by each job in $\mathbf{T}(P_L, \text{gcs, lower})$ at most $(\kappa_i + 1)$ times.

◆ Thus,

$$\text{LPD} \leq \sum_{T_k \in \mathbf{T}(P_L, \text{gcs, remote})} \left(\left\lfloor \frac{p_i}{p_k} \right\rfloor + 1 \right) c_{k, \text{total}}(P_L) + \sum_{T_k \in \mathbf{T}(P_L, \text{gcs, lower})} (\kappa_i + 1) c_{k, \text{max}}(P_i)$$

Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 15

Remote Blocking Time (RBT)

- ◆ Each time task T_i requests a global resource on a remote synchronization processor, its global critical section may be blocked once by a task that also requires some global resource and has a lower priority than π_i .

◆ Thus,

$$\text{RBT} \leq \sum_{j=1}^{\kappa_i} \max_{T_k \in \mathbf{T}(P_j, \text{gcs, lower})} (c_{k, \text{max}}(P_j))$$

Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 16

Remote Preemption Delay (RPD)

- ◆ RPD accounts for the amount of time task T_i is preempted by global critical sections of tasks in $\mathbf{T}(gcs, higher)$:
 - » During each period of T_i , each global cs of a task T_k in $\mathbf{T}(gcs, higher)$ may execute at most $\left\lfloor \frac{p_i}{p_k} \right\rfloor + 1$ times.

◆ Thus,

$$RPD \leq \sum_{T_k \in \mathbf{T}(gcs, higher)} \left(\left\lfloor \frac{p_i}{p_k} \right\rfloor + 1 \right) c_{k, total}$$

Deferred Blocking Time (DBT)

- ◆ DBT accounts for the total amount of time each job in task T_i can be blocked due to deferred execution of a local higher-priority task.
- ◆ Thus, $DBT \leq \sum_{T_k \in \mathbf{T}(local, higher)} \min(c_{k,L}, suspension_times_of_T_k)$
- ◆ Since *suspension_time_of_T_k* is often unknown, we simply bound this delay (even more loosely) by

$$DBT \leq \sum_{T_k \in \mathbf{T}(local, higher)} e_{k,L}$$

Total Blocking Time

- ◆ Thus, the total blocking time $b_i(rc)$ incurred by a job of task T_i is the sum of the previous five blocking terms:
 $b_i(rc) = LBT_i + LPD_i + RBT_i + RPD_i + DBT_i$
- ◆ See Sect 9.3.3 of Liu's text for a full example of calculating $b_i(rc)$.
- ◆ Observe that $b_i(rc)$ is a very loose upper bound on the total blocking time incurred by task T_i :
 - » The factors do not make use of many parameters of the tasks.
 - » The bound of each blocking factor assumes the worst-case blocking time.
- ◆ Tighter bounds can be derived for any given task set.

Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 19

Schedulability Tests for MPCP

We have already talked about how to incorporate blocking terms into scheduling conditions. We can determine the schedulability of each task T_i using the TDA or generalized TDA methods. The only change is that for each equal or higher priority task T_k , we exclude the total execution time of T_k 's global critical sections that are executed on remote processors where T_i does not execute.

This means we use $e_{k,L} + c_{k,total}$ rather than e_k for the execution time of task T_k . Thus, for TDA, we get this:

$$w_i(t) = e_i + b_i + \sum_{k=1}^{i-1} \left\lfloor \frac{t}{p_k} \right\rfloor \cdot (e_{k,L} + c_{k,total}) \quad \text{for } 0 < t \leq \min(D_i, p_i)$$

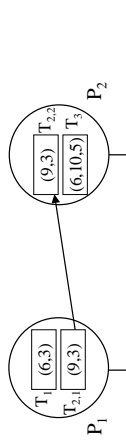
Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 20

Distributed Real-Time Systems

- ◆ In many distributed systems the workload can be modeled as a set of tasks in which some tasks consist of subtasks that execute on different processors:



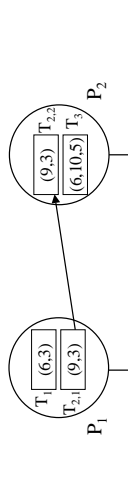
- ◆ The subtasks are released according to precedence constraints. That is, T_{2,2} cannot be released until task T_{2,1} completes.

End-To-End Scheduling

- ◆ The scheduling problem now becomes one of guaranteeing end-to-end response times in a distributed system with precedence constraints.
- ◆ We assume that each subtask (or task if it has no subtasks) accesses only local resources.
- ◆ Thus, we simplify the end-to-end scheduling problem by decomposing it to a set of similar scheduling problems that we have already discussed: single processor scheduling.
- ◆ We need only worry about scheduling tasks (with access to local resources) on a single processor and obeying precedence constraints.

End-to-End Scheduling (continued)

- ◆ We can use any scheduling algorithm we want as long as we can bound response times.
- ◆ We can even use a different algorithm on each processor.
- ◆ All we need is a synchronization protocol that ensures precedence constraints are met.



Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 23

Synchronization Protocols

- ◆ Work-conserving:
 - » Direct Synchronization (DS) Protocol
 - Liu calls this the Greedy Synchronization Protocol
 - Simple and low overhead, but may result in large release-time and/or finish-time jitter.
 - » Release-Guard (RG) Protocol (Sun) -- we cover this last
 - Combines MPM with DS.
- ◆ Non-work-conserving:
 - » Overcomes release/finish-time jitter by shaping the release-time patterns of subtasks.
 - » Non-work-conserving protocols are usually time-driven and require synchronized clocks (which creates new problems).
 - » Phase-Modification (PM) Protocol (Bettati)
 - » Modified Phase-Modification (MPM) Protocol

Steve Goddard

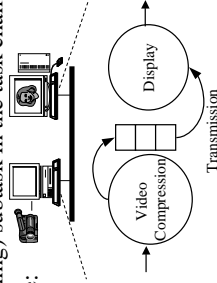
Real-Time Systems

Multiprocessor & Dist. Systems - 24

Direct Synchronization (DS) Protocol

- ◆ As soon as a subtask finishes executing, it sends a synchronization signal (e.g., message) to the “downstream” (sibling) subtask in the task chain.

- ◆ For example:



- ◆ Each subtask is released as soon as its predecessor completes, e.g., as soon as a message arrives.

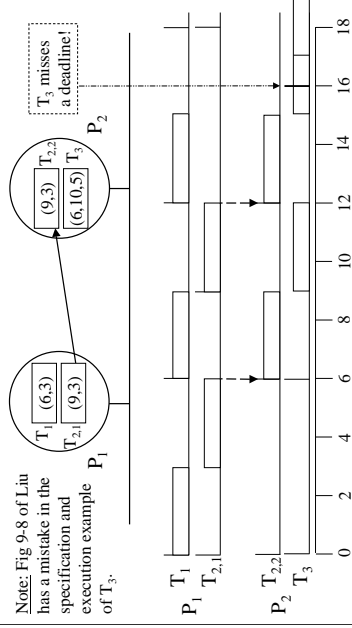
Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 25

RM Execution Example with DS

Note: Fig 9-8 of Liu has a mistake in the specification and execution example of T_3 .

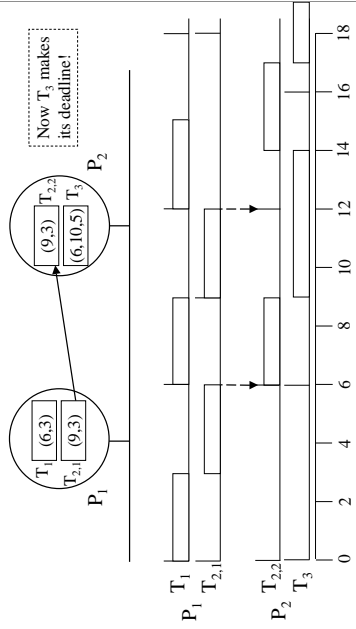


Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 26

EDF Execution Example with DS



Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 27

Advantages of DS Protocol

- ◆ It is the most commonly used protocol, especially in non-real-time systems.
- ◆ The DS protocol is simple and can be implemented in many ways.
- ◆ Global clock synchronization is not needed.
- ◆ Yields the shortest average end-to-end response time of all tasks when compared with the other synchronization protocols.

Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 28

Disadvantages of DS Protocol

- ◆ Some synchronized tasks may have extremely large end-to-end response times when scheduled with a fixed priority scheduler at each processor (even though this protocol yields the lowest average response time).
- ◆ The DS protocol often results in bursty release times of downstream subtasks.
 - » This can affect the schedulability of end-to-end tasks in fixed priority driven systems.
 - » The Rate-Based Execution Model was created to address this problem.
- ◆ It is difficult to assign priorities for subtasks on different processors in a priority-driven system.

Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 29

Phase-Modification (PM) Protocol

- ◆ The PM protocol maintains a minimum temporal distance between the release times of jobs in sibling subtasks of a task chain.
 - » The earliest release time of each subtask is some fixed amount of time after the release of immediate predecessor subtask in the task chain.
- ◆ Let $W_{i,k}$ be the maximum response time of every subtask $T_{i,k}$ on processor $V_{i,k}$. The j^{th} job $J_{i,k+1,j}$ of subtask $T_{i,k+1}$ is released on processor $V_{i,k+1}$ at time

$$\phi_i + (j-1)p_i + \sum_{l=1}^k W_{i,l}$$

Where the release time of the j^{th} job in $T_{i,1}$ is $\phi_i + (j-1)p_i$.

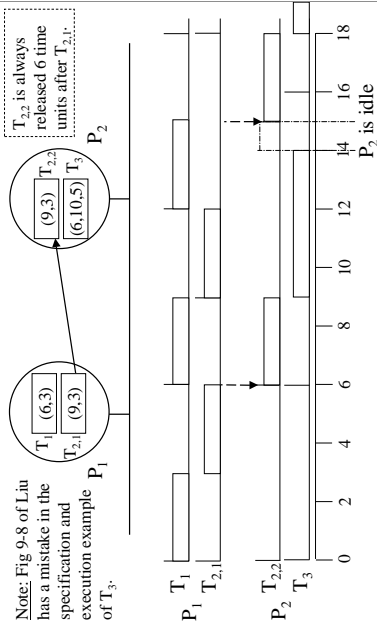
Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 30

RM Execution Example with PM

Note: Fig 9-8 of Liu has a mistake in the specification and execution example of T_3 .



Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 31

Advantages of PM Protocol

- ◆ Very simple concept.
- ◆ Easy to implement when
 - » clocks are synchronized,
 - » the first subtask of every task chain is released periodically, and
 - » $W_{i,k}$ of each subtask is available to the downstream scheduler.
- ◆ Each subtask is always released periodically if the first subtask is a periodic task.
- ◆ End-to-End response time is simply the sum

$$W_i = \sum_{l=1}^{m(i)} W_{i,l}$$

where $m(i)$ is the number of subtasks in task T_i .

Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 32

Disadvantages of PM Protocol

- ◆ Requires global clock synchronization and global information on upper bounds to response times.
- ◆ If there is any release-time jitter in the first subtasks, as is often the case, then the PM protocol may violate a precedence constraint.
- ◆ It is difficult to get tight (and precise) bounds on the maximum response time $W_{i,k}$ of each subtask.

Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 33

Modified Phase-Modification (MPM) Protocol

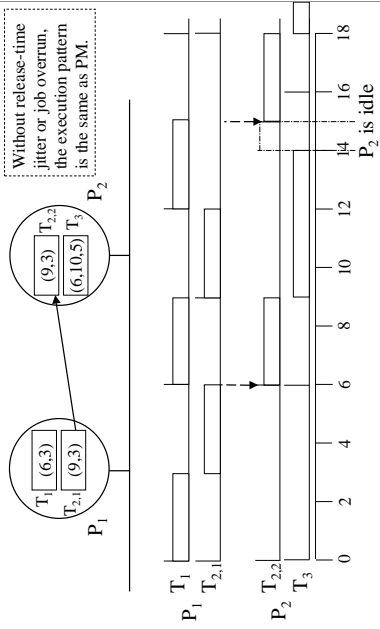
- ◆ The MPM protocol combines the synchronization message of DS with phase modification of PM.
- ◆ The only difference between PM and MPM is that the each successor job $J_{i,k+1,j}$ is released at the later of $W_{i,k}$ time units after the release of its predecessor job $J_{i,k,j}$ and the actual completion time of $J_{i,k,j}$.
- ◆ A simple implementation is for the scheduler of $V_{i,k}$ to send a synchronization signal to the scheduler of $V_{i,k+1}$ when the downstream job $J_{i,k+1,j}$ should be released.
 - » This way we do not need synchronized clocks, and
 - » release-time jitter of the first sub-task is propagated throughout the task chain to maintain precedence constraints.

Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 34

RM Execution Example with MPM



Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 35

Advantages of MPM Protocol

- ◆ Does not require a global clock!
- ◆ Easy to implement even when
 - » clocks are not synchronized, or
 - » the first subtask of every task chain is not released periodically.
- ◆ $W_{i,k}$ of each subtask is not needed by the downstream scheduler.
- ◆ Correct even with job overrun or release-time jitter (sporadic release times) for the first subtask in the task chain.

Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 36

Disadvantages of MPM Protocol

- ◆ The successor subtask $T_{i,k+1}$ may no longer behave like a periodic task when an upstream job overruns its wctt.
 - » Thus, the scheduling analysis on two processors may be incorrect!
- ◆ The non-work-conserving nature of the protocol increases average response times unnecessarily when the downstream processor is idle.
 - » This was also true with PM.

Release-Guard (RG) Protocol

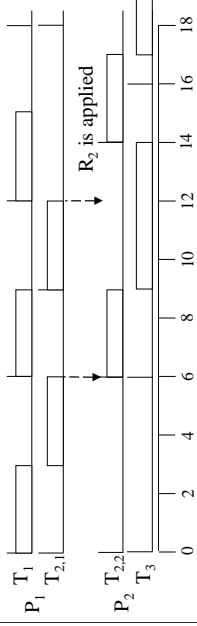
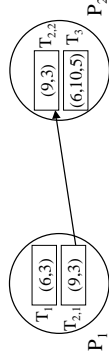
- ◆ Let $r_{i,k,j}$ be the release time of job $J_{i,k,j}$ on processor $V_{i,k,j}$.
- ◆ The scheduler of $V_{i,k}$ sends a synchronization signal to the scheduler of $V_{i,k+1}$ as soon as job $J_{i,k,j}$ completes.
- ◆ The scheduler of $V_{i,k+1}$ releases the first job $J_{i,k+1,l}$ of subtask $T_{i,k}$ when it receives the first synchronization signal for the subtask.
- ◆ For $j > 1$, if the processor is busy in the interval $[r_{i,k+1,j-1}, r_{i,k+1,j-1} + p_i]$, R1 job $J_{i,k+1,j}$ is released at the later of when the synchronization signal is received from the predecessor job $J_{i,k,j}$ and $r_{i,k+1,j-1} + p_i$.
- ◆ Otherwise,
 - R2 for $j > 1$, job $J_{i,k+1,j}$ is released at the later of when the synchronization signal is received from the predecessor job $J_{i,k,j}$ and when the processor becomes idle.

Intuition Behind (RG) Protocol

- ◆ Intuitively, the RG protocol makes sure that the inter-release time of any two consecutive jobs of the same subtask are never less than the period of the subtask.
- ◆ But it will release jobs early when doing so will not affect the schedulability of lower-priority subtasks.

RM Execution Example with RG

R2 is applied at time 14 to release $T_{2,2}$ early since P_2 would otherwise be idle.



Advantages of RG Protocol

- ◆ Work-conserving:
 - » Takes advantage of idle time to reduce end-to-end response times.
- ◆ Does not require a global clock!
- ◆ $W_{i,k}$ of each subtask is not needed by the downstream scheduler.
- ◆ Correct even with job overrun or release-time jitter (sporadic release times) for the first subtask in the task chain.
- ◆ Liu gives no disadvantages!
 - » Can you think of any?

Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 41

Relative Performance of the Four Protocols

- ◆ The DS has low overhead/complexity and the shortest average end-to-end response (EER) times.
 - » However, it can result in very long worst-case EER times.
- ◆ The RG protocol yields better average EER than PM or MPM.
 - » However, output jitter is much worse than that produced by PM or MPM.
 - » Liu indicates that if output jitter is a concern, you can use RG and simply ignore rule R2.
- ◆ Liu contends that RG is almost always better than DS in actual systems because of the shorter worst-case EER times RG yields.

Steve Goddard

Real-Time Systems

Multiprocessor & Dist. Systems - 42