Dynamic Scheduling of Parallel Real-time Jobs by Modelling Spare Capabilities in Heterogeneous Clusters

Ligang He, Stephen A. Jarvis, Daniel P. Spooner and Graham R. Nudd

High Performance System Group,

Department of Computer Science,

University of Warwick

Coventry, UK, CV4 7AL

Presented by Ligang He

Motivation

- Reservation of real time computations in resources
- New real time jobs keep arriving at the resources for computation
- Scheduling newly arriving real time jobs to resources while the reserved real time computations are still guaranteed

Objective

- Modeling the spare capabilities left by the existing periodic real time jobs in heterogeneous cluster
- Scheduling parallel real-time jobs with the topology of Directed Acyclic Graph (DAG), based on the modeling approach for spare capabilities

Contributions

- An optimal approach is presented to model the spare capabilities left by periodic jobs in a heterogeneous cluster
- A dynamic scheduling mechanism is proposed to satisfy the real-time requirements of both existing jobs and newly arriving jobs

Presentation Outline

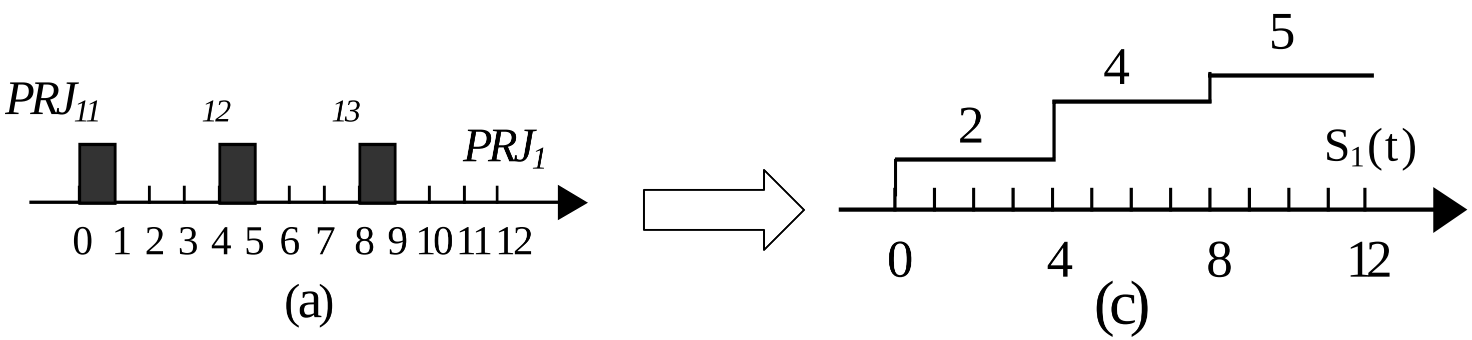
- Modeling spare capabilities in a heterogeneous cluster
- Scheduling of parallel real-time jobs with DAG topology
- Experimental studies

Modeling spare capabilities in a heterogeneous cluster

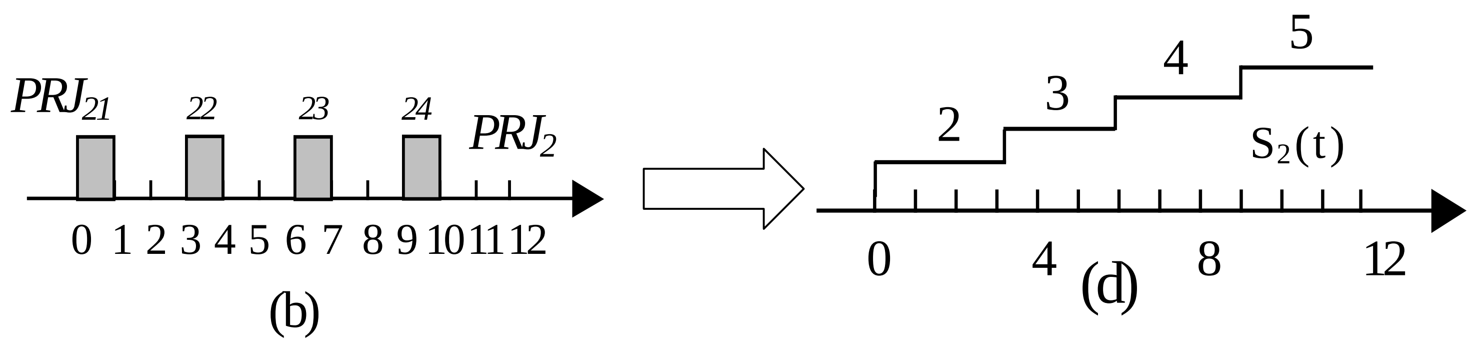
Difficulties

- In the cluster architecture, the global scheduler usually locates in a centric computer while the jobs are run in other processing computers
- The global scheduler has to model the spare capabilities of other computers rather than a processing computer models the spare capability in itself

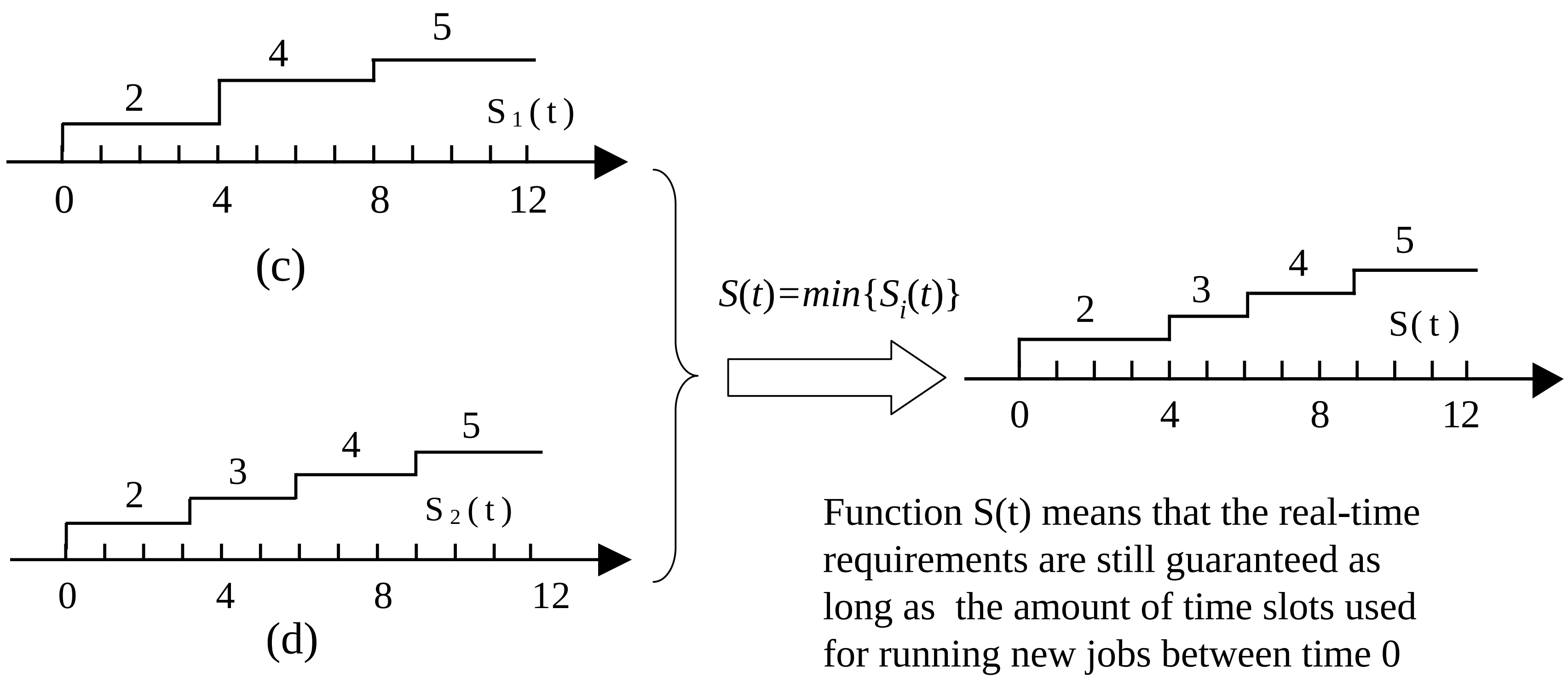
Function of Spare time slots (1)



$$S_i(t) = D_{ij} - P_{ij} \quad D_{i(j-1)} < t \le D_{ij}$$



Function of Spare time slots (2)



Function S(t) means that the real-time requirements are still guaranteed as long as the amount of time slots used for running new jobs between time 0 and any time point does not exceed the value indicated by the function

Modeling Spare Capabilities on-line

- Function S(t) can be constructed off-line for every computer in the heterogeneous cluster
- Function S(t) indicates the time slots available for new jobs between time 0 and any time point in the future.
- The dynamic arrivals of new jobs complicate the problem since their arriving times may not be 0.
- Suppose a new job arrives at time t_0, we need to work out on-line how many time slots are available for the job between its arrival time t_0 and a time point t after t_0, denoted by $S(t_0, t)$

Compute $S(t_0, t)$ (Theorem 1 in the paper)

- W_1: work amount to be finished before t
  - sum of exec times of instances whose deadlines are less than t
- W_2: work amount having been finished before t_0
- W_1 minus W_2 is the work amount to be finished in $[t_0, t]$
- The left time slots are spare, that is,

$$S(t_0, t)=(t-t_0)-(W_1-W_2)=(t-W_1)-t_0+W_2$$

- Since $t-W_1=S(t)$, we just need to work out W_2

Compute W_2

- W_2 includes two parts
  - W_{2a}: the sum of exec times of instances whose deadlines are less than t_0
  - W_{2b}: work finished before t_0 for the instances whose deadlines are greater than t_0 but less than t
- W_{2a} is easy to compute, the problem is narrowed down to work out W_{2b}

Compute W_{2b}

- The instances whose deadlines are greater than t_0 can only occupy the time slots left by the instances whose deadlines are less than t_0 (because of EDF policy)
- To compute W_{2b}, we need to work out the distribution pattern of time slots left by the instances with deadlines less than t_0

Distribution of time slots left by instances with deadlines less than t_0

- If only existing periodic jobs are running, the distribution of time slots is easy to work out (which can be done off-line)
- However, the arrival and execution of previous new tasks will disturb the original distribution
- A property is revealed about how the previous new tasks disturb the original distribution

A property (Theorem 2 in the paper)

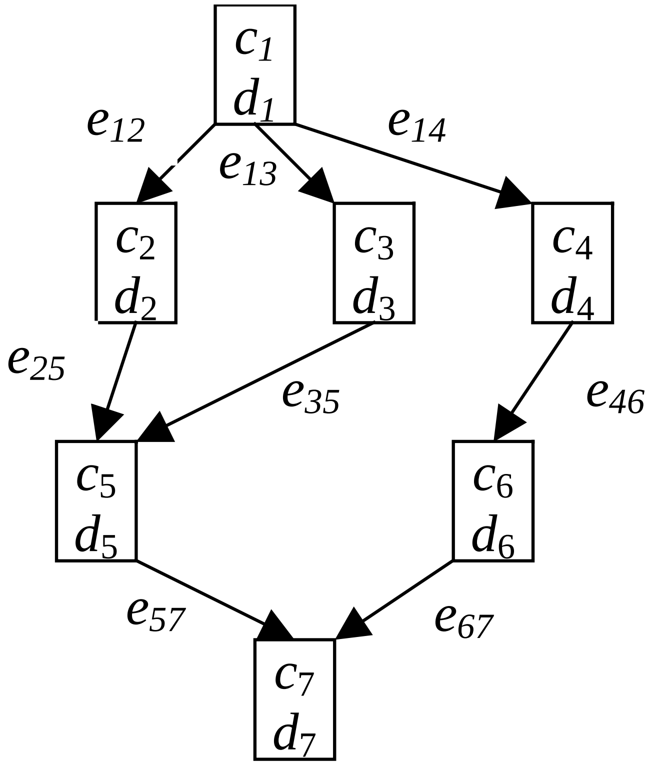
- Suppose the last executed new task is completed at time f, then there exists such a time point t_s in $[f, t_0]$, that
  - The instances with deadlines less than t_0 retain the original execution pattern in $[t_s, t_0]$ as if there were no previous new tasks run before
  - There are no idle time slots in $[f, t_s]$
- Since the original pattern is retained, W_{2b} can be worked out. Hence $S(t_0, t)$ can be computed.

Advantage of the model approach

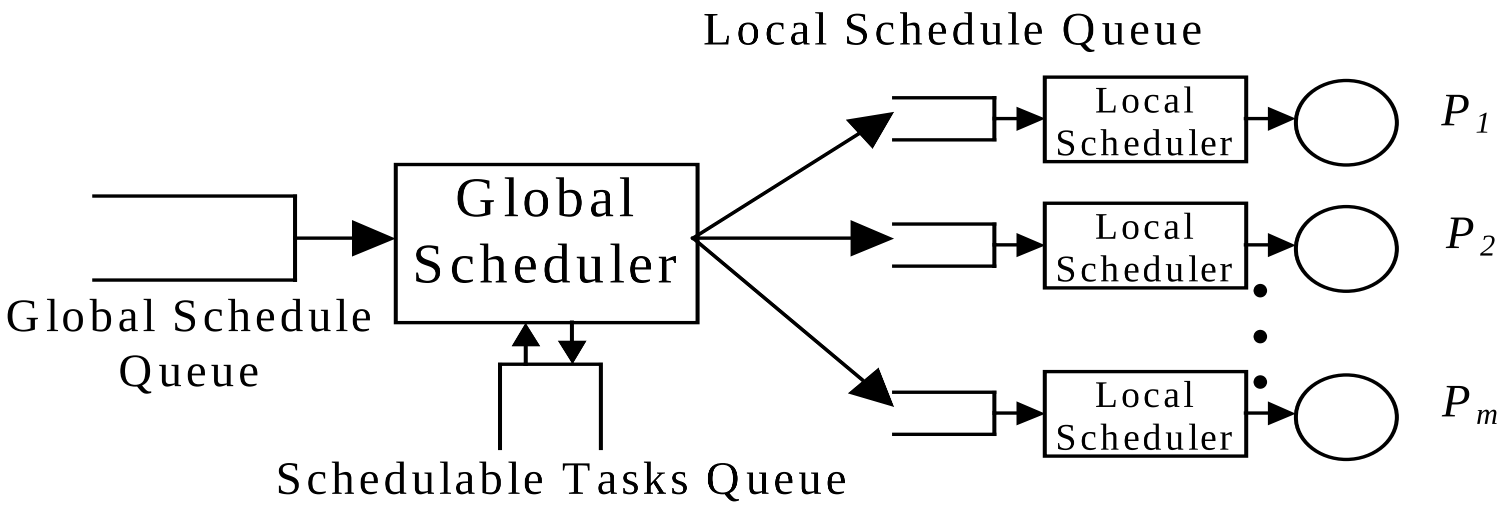
- The modeling procedure obtains the maximal spare time slot available for running new jobs between the job's arrival time and a future time point, so that new jobs can achieve the optimal response time
- Free of communication between the global scheduler and other processing computers in the modeling procedure

Scheduling of parallel real-time jobs with DAG topology

The Model of Parallel Real-time Jobs



The Scheduler Model (1)



The scheduling procedure (1)

- There are existing periodic jobs running in processing computers
- New parallel real time jobs arrive at the global scheduler for execution
- Each time the global scheduler gets a job from the global schedule queue and searches for the schedulable subtasks in the job and put them into the schedulable tasks queue, then the global scheduler get a subtask from the schedulable tasks queue and schedule it onto one of processing computers
- When deciding which computer to choose, The global scheduler will compute the spare capability in each computer
- In each processing computer, the local scheduler schedules the tasks in the local schedule queue as the EDF policy (the task is a subtask of a new parallel job or a periodic job instance)

Compute the finish time of a task (Algorithm 1 in the paper)

- 1. Get a time point t_k after t_0 that $S(t_k)$ changes
- 2. Compute $S(t_0, t_k)$
- 3. If $S(t_0, t_k)$ < task's execution time c_i
- 4. Get $k=k+1$, go to Step 2
- The task's finish time is

$$t_{k-1}+c_i-S(t_0, t_{k-1})$$

Admission Control (Algorithm 2 in the paper)

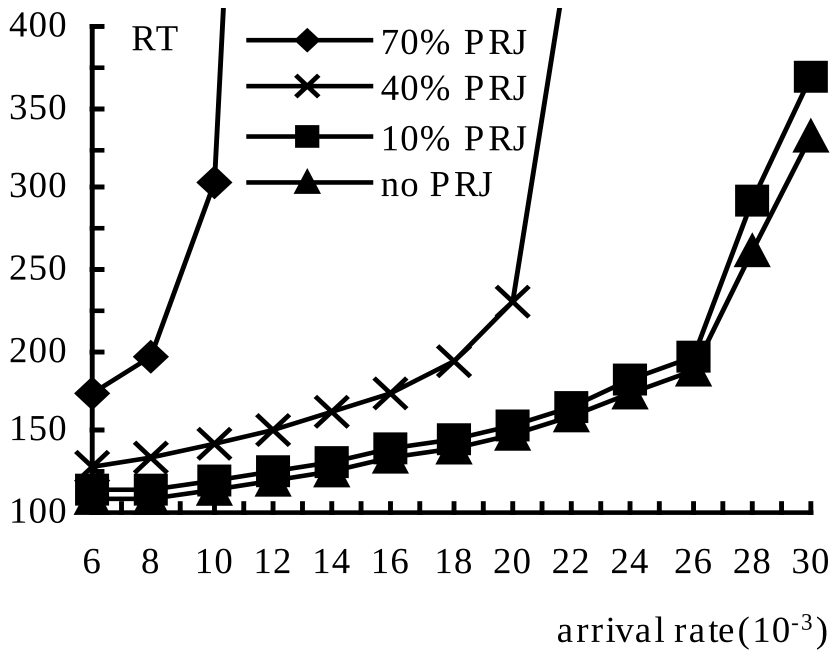
- If the finish time of a task in any processing computer is greater than its deadline, the parallel real-time job that the task belongs to is rejected
- If multiple computers can satisfy the task's deadline, two possible selection policies are applied
  - Select the computer which provides the shortest finish time (Response-First policy)
  - Select the computer which provides the longest finish time (Utilization-First policy)
- After deciding which computer the task should be sent to, the deadline of the task is reset to be the finish time of the task in that computer

Advantage of the scheduling procedure

- Since the computed spare capabilities are maximal possible, the computed finish time of a task is shortest possible while the real time requirements of all periodic jobs are still guaranteed

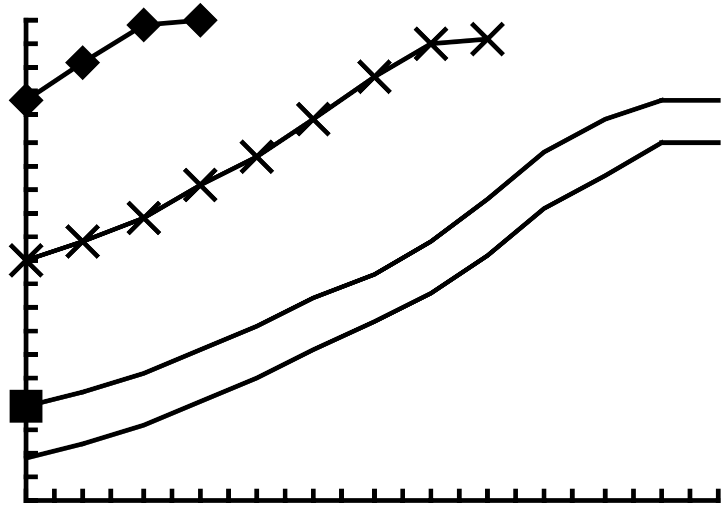
Experimental Studies

Job workload (1)

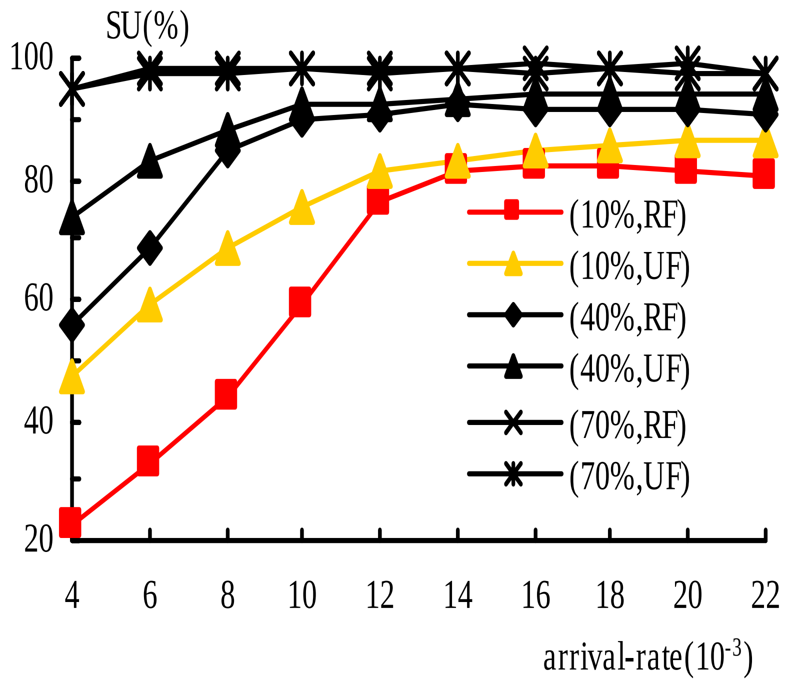


Effect of workload on average response time under different levels of PRJs

Job workload (2)

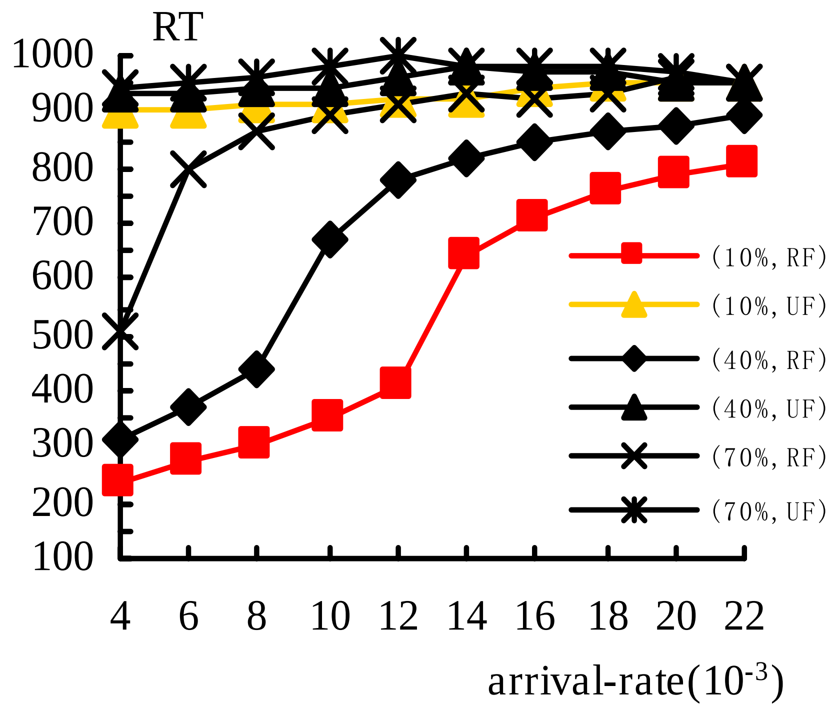


Selection Policies (RF vs UF) (1)



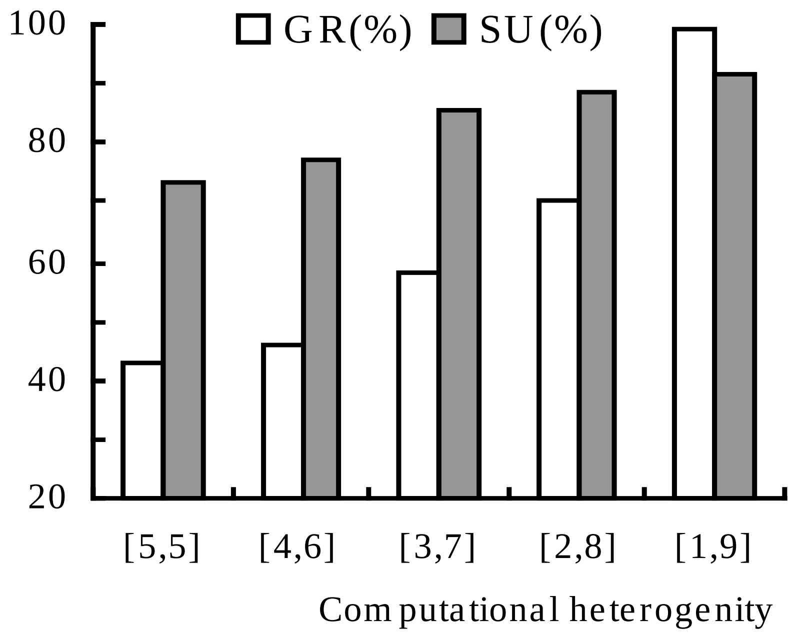
Effect of Job workloads and second-level selection policy on SU

Selection Policies (RF vs UF) (2)



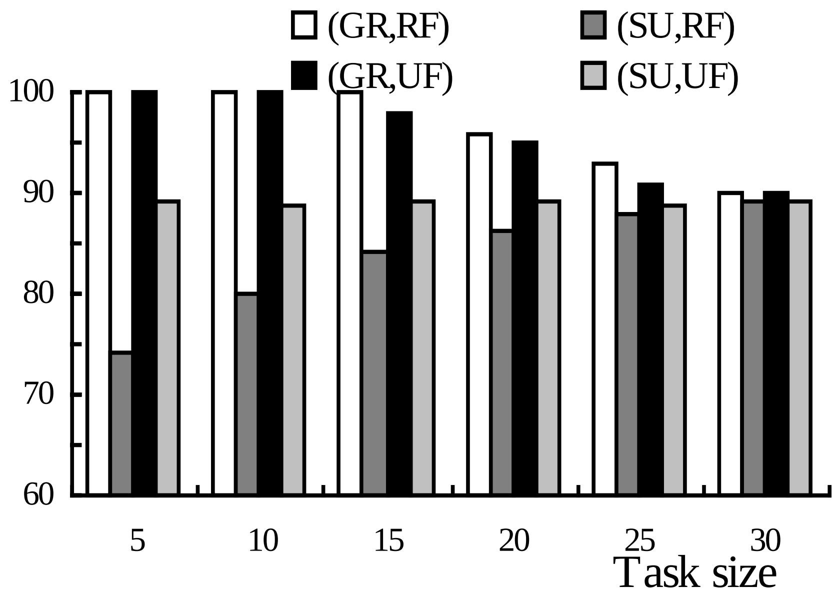
Effect of Job workloads and second-level selection policy on RT

Computation heterogeneity



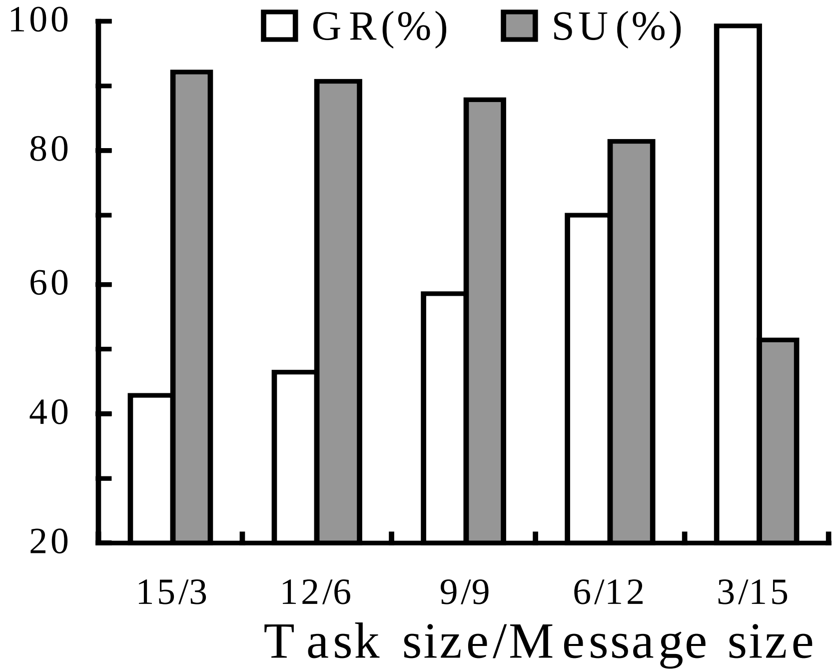
Effect of computation heterogeneity on GR and SU

Task size



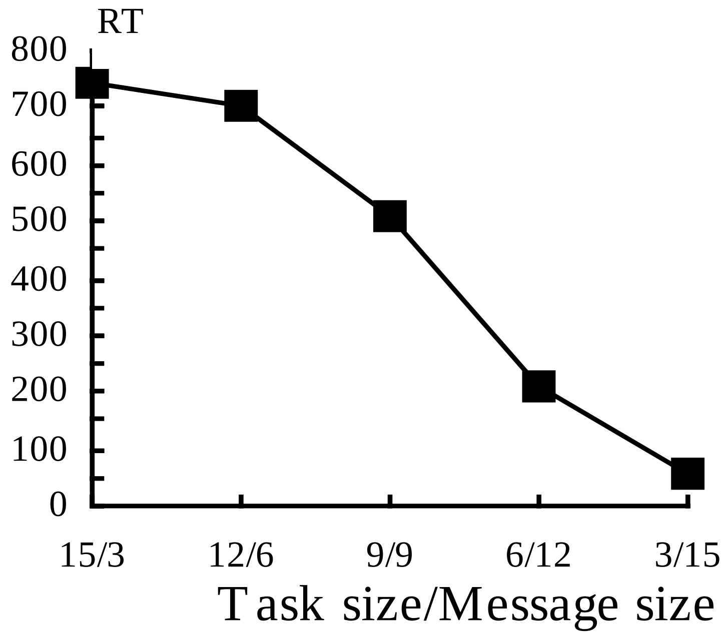
Effect of task size on GR and SU

Task size and Message size (1)



Effect of task-size/message-size ratio on GR and SU

Task size and Message size (2)



Effect of task-size/message-size ratio on RT

Conclusions

- An optimal approach for modeling the spare capabilities in a cluster left by the periodic jobs
- A dynamic scheduling mechanism is developed to satisfy the real-time requirements of both existing periodic jobs and newly arriving parallel real-time jobs