Computation Efficiency Driven Job Removal Policies for Meeting End-to-End Deadlines in Distributed Real-Time Systems

Miao Song†, Shuhui Li†, Shangping Ren†
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois 60616
Email: {msong8, sli38, ren}@iit.edu

Shengyan Hong‡, Xiaobo Sharon Hu‡
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556
Email: {shong3, shu}@nd.edu

***Abstract*—In distributed real-time systems, when resource cannot meet workload demand, some jobs have to be removed from further execution. The decision as to which job to remove directly influences the system computation efficiency, i.e., the ratio between computation contributed to successful completions of real-time jobs and total computation contributed to the execution of jobs that may or may not be completed. The paper presents two job removal policies which aim at maximizing system's computation efficiency for distributed real-time applications where the applications' end-to-end deadlines must be guaranteed. Experiments based on benchmark applications generated by TGFF [1] are conducted and compared with recent work in the literature. The results show clear benefits of the developed approaches — they can achieve as much as 20% computation efficiency improvement.**

I. Introduction

In distributed real-time systems, such as vehicle control and multimedia communication systems [2], [3], jobs are usually required to be executed on a chain of processors and be completed within given end-to-end deadlines. As jobs are executed on distributed processors, local deadlines need to be assigned to jobs when they arrive at each processor so that their end-to-end deadlines can be met. When not all local deadline constraints of jobs on a processor can be satisfied, some jobs need to be removed from the processor's execution queue so that the remaining jobs can meet their end-to-end deadlines.

Different job removal criteria are proposed in the past. In [4], [5], job removals are based on their the criticality levels: the low critical jobs are removed to guarantee the schedulability of all the high critical jobs. In [6], job removal is aimed at minimizing the weighted number of jobs and weighted completion time. Yang et al. [7] remove jobs by considering temperature constraints. A recently proposed job removal policy [8] aims to minimize the number of jobs to be removed. In particular, it removes the job that has the longest unfinished execution time. When a job with the largest future computation demand is removed from future execution, the remaining jobs can share the maximum computation time savings for their later stages and can hence have better chances to meet their end-to-end deadlines. Therefore, the number of jobs removed tends to be small.

Unfortunately, none of the job removal criteria has taken into consideration of system computation efficiency, i.e., the ratio between computation contributed to successful job completions and total computation carried out by the system. It is not difficult to see that the ratio is directly related to system's energy consumption efficiency — performing fruitless computation is a waste of energy.

In recent years, the increased amount of energy consumption caused by computer systems is no longer negligible to nation's power plants. In 2006, approximately 61.4 billion KWh was consumed by data centers alone in the United States and it was around 1.2% of the total U.S. energy consumption. According to [9], [10], this percentage doubles every five years. More recent data shows that in 2010, 408 TWhs was consumed by computers worldwide with 10% annual increase on installed computers [11]. Hence, it is imperative that when we design computer systems, we must take computation efficiency into consideration, specially for real-time applications as they are more often to have limited energy resources.

With priority based scheduling algorithms, such as the EDF algorithm, job removal procedure works hand-in-hand with the procedure that assigns local deadlines to distributed applications. The removal decision is made based on local information a processor has, such as the workload on the processor. However, such information may not be known a priori on processors where job arrivals are not known and could be at arbitrary time. Therefore, we focus on online job removal.

In this paper, we propose two computation efficiency driven heuristic job removal policies for meeting end-to-end deadlines of distributed real-time applications. We compare and analyze the advantages and disadvantages of the proposed two approaches. Extensive simulations based on job models generated by TGFF ("Task Graphs For Free") [1] are con-

† The research is supported in part by NSF under grant number CAREER 0746643 and CNS 1018731.

‡ The research is supported in part by NSF under grant number CPS-0931195 and a research contract from the Sandia National Laboratories.

ducted to verify the benefits of the proposed policies over an existing job removal policy. Our experimental results show that the proposed computation efficiency driven job removal policies not only result in high computation efficiency, but also maintain reasonably low job removal ratio.

The rest of the paper is organized as follows. In Section II, we formulate the research problem the paper is to address. It is followed by background information regarding local deadline assignment criteria for distributed real-time applications with end-to-end deadlines (Section III). Two new computation efficiency driven job removal policies, i.e., the Least Completion ratio First (LCF) and the Maximum Potential computation efficiency First (MPF) job removal policies, are presented in Section IV. Experimental results are discussed in Section V. Finally, we conclude the paper in Section VI.

II. Problem Formulation

In this section, we first introduce the models and definitions and then formulate the research problem which the paper is to address.

Processor Model

The distributed system in our context consists of p networked processing units, denoted as $\mathcal{P} = \{V_1, V_2, \cdots, V_p\}$. The job set on each individual processors is scheduled based on the preemptive EDF scheduling algorithm. We further assume that the energy cost per unit of computation is the same among all the processors, and communication time and the related energy cost are negligible.

Application Model

Real-time applications are modeled as a set of independent jobs denoted as $\Gamma = \{J_1, J_2, \cdots, J_n\}$. Each job has an end-to-end deadline and may go through a chain of processors. More precisely, a real-time job J_i is represented by a 3-tuple, i.e., $J_i = (D_i, E_i, P_i^{s_i}[(V_x, J_{i,j})])$, where

- D_i: end-to-end deadline;
- E_i: end-to-end execution time;
- $P_i^{s_i}[(V_x, J_{i,j})]$: job execution path, where V_x is the processor on which job J_i's jth stage, i.e., $J_{i,j}$, is executed;
- s_i: the length of job J_i's execution path. We assume a job does not repeat on a processor, i.e., $s_i \leq p$;
- $J_{i,j}$: the jth sub-job of J_i, $1 \leq j \leq s_i$. $J_{i,j} = (r_{i,j}, e_{i,j}, d_{i,j})$;
- $r_{i,j}$: release time of job J_i at its jth stage. We assume sub-job $J_{i,j}$ is released as soon as its predecessor sub-job $J_{i,j-1}$ is finished;
- $e_{i,j}$: execution time of job J_i at its jth stage, $\sum_{j=1}^{s_i} e_{i,j} = E_i$;
- $d_{i,j}$: local deadline assigned to $J_{i,j}$.

We use jth stage, or jth sub-job interchangeably in the paper.

Definitions

Definition 1: **Average processor utilization ($\mathcal{A}$)**: given a set of processors $\mathcal{P} = \{V_1, \cdots, V_p\}$ and a job set $\Gamma = \{J_1, \cdots, J_n\}$, where $J_i = (D_i, E_i, P_i^{s_i}[(V_x, J_{i,j})])$, the system average processor utilization is defined as:

$$\mathcal{A} = \frac{\sum_{i=1}^{n} \frac{E_i}{D_i}}{p} \tag{1}$$

□

Definition 2: **Stage indicator** $k(i,x)$: for $J_i \in \Gamma$ and $V_x \in \mathcal{P}$, if a sub-job of J_i is executed on processor V_x, function $k(i,x)$ returns a stage index j, where $j = k(i,x)$ indicates that job J_i's jth stage, i.e., sub-job $J_{i,j}$, is executed on processor V_x. □

Definition 3: **Job removal ratio** $\gamma(t)$: within time duration $[0,t]$, if the number of jobs released is N_r, and the number of jobs removed before they are successfully completed is N_d, the job removal ratio is defined as: $\gamma(t) = N_d/N_r$. □

Definition 4: **Current stage accrued execution time** $C_{\text{cur}}(J_{i,j}, t)$: time duration that sub-job $J_{i,j}$ is executed before t at stage j and $0 \leq C_{\text{cur}}(J_{i,j}, t) \leq e_{i,j}$. □

Definition 5: **Total accrued execution time** $C_{\text{tot}}(J_{i,j}, t)$: time duration that job J_i is executed within time duration $[0,t]$. If at time t, job J_i is at its jth stage, $C_{\text{tot}}(J_{i,j}, t) = \sum_{m=1}^{j-1} e_{i,m} + C_{\text{cur}}(J_{i,j}, t)$. □

Definition 6: **Remaining execution time** $C_{\text{rem}}(J_{i,j}, t)$: if at time t, job J_i is at its jth stage, the remaining execution time of J_i is $C_{\text{rem}}(J_{i,j}, t) = E_i - C_{\text{tot}}(J_{i,j}, t)$. □

Definition 7: **Job execution completion ratio** $\eta(J_{i,j}, t)$: if job J_i is at its jth stage at time t, its execution completion ratio is defined as: $\eta(J_{i,j}, t) = C_{\text{tot}}(J_{i,j}, t)/E_i$, where E_i is the end-to-end execution time of J_i. □

Definition 8: **Effective computation time** $E_s(t)$: computation time contributed to successful completion of jobs before their end-to-end deadlines within time duration $[0,t]$. Let Γ_{succ} denote the job set that are successfully completed before their end-to-end deadlines within $[0,t]$, then

$$E_s(t) = \sum_{J_i \in \Gamma_{\text{succ}}} E_i \tag{2}$$

where $\Gamma_{\text{succ}} \subseteq \Gamma$. □

Definition 9: **Wasted computation time** $E_f(t)$: computation time contributed to executing jobs that are removed before their completion within time duration $[0,t]$. Let Γ_{fail} denote the jobs that are removed before their completion and t_i represents the time point at which job $J_{i,j}$ is removed,

$$E_f(t) = \sum_{J_i \in \Gamma_{\text{fail}} \wedge 0 \leq t_i \leq t} C_{\text{tot}}(J_{i,j}, t_i) \tag{3}$$

where $\Gamma_{\text{fail}} \subseteq \Gamma$. □

Definition 10: **System potential computation efficiency** $\rho(J_{i,j}, t)$: assume after job $J_{i,j} \in V_x$ is removed from V_x at time t, the remaining jobs then have the potential to complete successfully, the computation ratio defined by (4) is called potential computation efficiency.

$$\rho(J_{i,j}, t) = \frac{\sum_{J_h \in \Omega(V_x) \wedge h \neq i} E_h}{C_{\text{tot}}(J_{i,k(i,x)}, t) + \sum_{J_h \in \Omega(V_x) \wedge h \neq i} E_h} \tag{4}$$

where $\Omega(V_x)$ is the sub-job set on V_x at time t. □

Definition 11: **System computation efficiency** $\mu(t)$: within time duration $[0, t]$, system computation efficiency is defined as

$$\mu(t) = \frac{E_s(t)}{E_f(t) + E_s(t)} = 1 - \frac{E_f(t)}{E_f(t) + E_s(t)} \tag{5}$$

□

Research problem

When the system resource cannot meet workload demand and job removal is unavoidable, it is important to judiciously decide which job to remove so that the system's computation efficiency is maximized. The problem is formulated as follows.

Problem 1: given a set of networked processors $\mathcal{P}$ and a set of independent real-time jobs Γ as defined before, develop an online job removal policy that maximizes the system computation efficiency within time duration $[0, t]$, i.e.

$$\max \mu(t) \tag{6}$$

□

III. Background and Motivation

As online job removal policy heavily depends on processor's scheduling algorithm and local deadlines assigned to sub-jobs, we first briefly summarize the schedulability conditions for EDF based scheduling algorithm.

Given a job set Γ and a processor set $\mathcal{P}$ as defined before, let $\Omega(V_x)$ denote the sub-job set to be scheduled on processor V_x at a scheduling point, where $V_x \in \mathcal{P}$, $\Omega(V_x) \subseteq \Gamma$. Assume a local deadline assignment algorithm assigns $d_{i,k(i,x)}$ as $J_{i,k(i,x)}$'s local deadline on processor V_x, where $J_{i,k(i,x)} \in \Omega(V_x)$. If $\Omega(V_x)$ is schedulable under EDF scheduling algorithm, then we have: $\forall V_x \in \mathcal{P} \wedge \forall J_{i,k(i,x)}, J_{l,k(l,x)} \in \Omega(V_x)$

$$r_{i,k(i,x)} + e_{i,k(i,x)} \leq d_{i,k(i,x)} \leq D_i - \sum_{m=k(i,x)+1}^{s_i} e_{i,m} \tag{7}$$

$$d_{i,k(i,x)} - r_{l,k(l,x)} \leq \sum_{\substack{J_{h,k(h,x)} \in \Omega(V_x), \\ r_{h,k(h,x)} \geq r_{l,k(l,x)}, \\ d_{h,k(h,x)} \leq d_{i,k(i,x)}}} e_{h,k(h,x)} \tag{8}$$

Formula (7) bounds the local deadline of each sub-job on V_x between its earliest completion time at current stage and the latest start time of its immediate next stage. Formula (8) is the schedulability constraint for jobs on $\Omega(V_x)$ to be schedulable by EDF [12], [13].

When the two constraints cannot be both satisfied, some jobs in $\Omega(V_x)$ need to be removed from further execution. We take a recently developed local deadline assignment algorithm, the OLDA algorithm [8], as an example to show how a job removal policy works together with the EDF scheduling algorithm based on local deadlines assigned by the OLDA algorithm to meet end-to-end deadlines.

Example 1: Assume a system has three processors V_1, V_2, V_3 and three jobs J_1, J_2, J_3 whose execution information is given below and also shown in Fig. 1:

$$J_1 = \{71, 64, P_1^3[(V_1, (0, 30, ?)), (V_2, (?, 22, ?)), (V_3, (?, 12, ?))]\}$$
$$J_2 = \{77, 30, P_2^2[(V_1, (0, 7, ?)), (V_2, (?, 23, ?))]\}$$
$$J_3 = \{72, 11, P_3^2[(V_1, (0, 3, ?)), (V_2, (?, 8, ?))]\}$$

where the question marks (?) denote information to be decided by the OLDA algorithm.

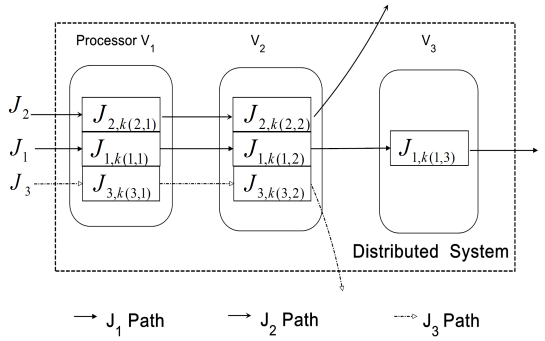


Fig. 1. Job Execution Paths

At $t = 0$, three jobs are all released to the first processor. By applying the OLDA algorithm, we obtain local deadlines for $J_{1,k(1,1)}$, $J_{2,k(2,1)}$ and $J_{3,k(3,1)}$ to be 30, 37 and 40, respectively. Hence, at time $t = 0$, the job information we have is:

$$J_1 = \{71, 64, P_1^3[(V_1, (0, 30, 30)), (V_2, (?, 22, ?)), (V_3, (?, 12, ?))]\}$$
$$J_2 = \{77, 30, P_2^2[(V_1, (0, 7, 37)), (V_2, (?, 23, ?))]\}$$
$$J_3 = \{72, 11, P_3^2[(V_1, (0, 3, 40)), (V_2, (?, 8, ?))]\}$$

Based on EDF scheduling policy, job $J_{1,k(1,1)}$ is executed first. At $t = 30$, it finishes its execution on V_1, and $J_{1,k(1,2)}$ is released to V_2, which triggers the OLDA to assign local deadlines for jobs on V_2. In other words, at $t = 30$, we have:

$$J_1 = \{71, 64, P_1^3[(V_1, (0, 30, 30)), (V_2, (30, 22, 52)), (V_3, (?, 12, ?))]\}$$
$$J_2 = \{77, 30, P_2^2[(V_1, (0, 7, 37)), (V_2, (?, 23, ?))]\}$$
$$J_3 = \{72, 11, P_3^2[(V_1, (0, 3, 40)), (V_2, (?, 8, ?))]\}$$

At $t = 37$, $J_{2,k(2,1)}$ finishes its execution on V_1, and $J_{2,k(2,2)}$ is released to V_2. By this time, $J_{1,k(1,2)}$ has executed 7 time units on V_2. The OLDA assigns the local deadline as $d_{2,k(2,2)} = 75$.

$$J_1 = \{71, 64, P_1^3[(V_1, (0, 30, 30)), (V_2, (30, 22, 52)), (V_3, (?, 12, ?))]\}$$
$$J_2 = \{77, 30, P_2^2[(V_1, (0, 7, 37)), (V_2, (37, 23, 75))]\}$$
$$J_3 = \{72, 11, P_3^2[(V_1, (0, 3, 40)), (V_2, (?, 8, ?))]\}$$

At $t = 40$, $J_{3,k(3,1)}$ finishes its execution on V_1, and $J_{3,k(3,2)}$ is released to V_2, new local deadlines need to be reassigned to the current sub-job set on V_2. By this time, $J_{1,k(1,2)}$ has executed 10 time units on V_2 and $J_{2,k(2,2)}$ does

not have a chance to start on V_2 yet. The OLDA assigns local deadline as: $d_{1,k(1,2)} = 52$, $d_{2,k(2,2)} = 83$ and $d_{3,k(3,2)} = 60$.

$$J_1 = \{71, 64, P_1^3[(V_1, (0, 30, 30)), (V_2, (30, 22, 52)), (V_3, (?, 12, ?))]\}$$
$$J_2 = \{77, 30, P_2^2[(V_1, (0, 7, 37)), (V_2, (37, 23, 83))]\}$$
$$J_3 = \{72, 11, P_3^2[(V_1, (0, 3, 40)), (V_2, (40, 8, 60))]\}$$

However, the end-to-end deadline of J_2 is 77, which is earlier than $d_{2,k(2,2)} = 83$ — violating constraint (7). Hence, $\Omega(V_2)$ at $t = 40$ is not schedulable and some job(s) need to be removed so that the remaining jobs can meet their end-to-end deadlines.

If we remove the job with the maximum remaining execution time as it is done in [8], then as J_1 has $12 + 12 = 24$ remaining time units, larger than what J_2 and J_3 have (which are 23 and 8, respectively), sub-job $J_{1,k(1,2)}$ is removed. $J_{2,k(2,2)}$, $J_{3,k(3,2)}$ are assigned with new local deadlines as below:

$$J_2 = \{77, 30, P_2^2[(V_1, (0, 7, 37)), (V_2, (37, 23, 71))]\}$$
$$J_3 = \{72, 11, P_3^2[(V_1, (0, 3, 40)), (V_2, (40, 8, 48))]\}$$

Both J_2 and J_3 are able to complete before their deadlines and J_1 is removed after it is executed for 40 time units. Therefore, within time duration of $[0, 77]$, the system's computation efficiency is $\mu(77) = 1 - \frac{40}{40+30+11} = 51\%$, about 49% of the computation performed is wasted.

However, at time $t = 40$, if we remove $J_{2,k(2,2)}$ instead of $J_{1,k(1,2)}$ from executing on V_2. We have

$$J_1 = \{71, 64, P_1^3[(V_1, (0, 30, 30)), (V_2, (30, 22, 52)), (V_3, (?, 12, ?))]\}$$
$$J_3 = \{72, 11, P_3^2[(V_1, (0, 3, 40)), (V_2, (40, 8, 60))]\}$$

At time $t = 52$, $J_{1,k(1,2)}$ finishes its execution, $J_{1,k(1,3)}$ is thus released to V_3. We have

$$J_1 = \{71, 64, P_1^3[(V_1, (0, 30, 30)), (V_2, (30, 22, 52)), (V_3, (52, 12, 64))]\}$$
$$J_3 = \{72, 11, P_3^2[(V_1, (0, 3, 40)), (V_2, (40, 8, 60))]\}$$

$J_{1,k(1,3)}$ and $J_{3,k(3,2)}$ finish their execution at 64 and 60, respectively. Only J_2 is removed with 7 time units wasted. Hence, within time duration of $[0, 77]$, the system's computation efficiency is $\mu(77) = 1 - \frac{7}{64+7+11} = 91\%$.

At $t = 40$, the third choice is to remove $J_{3,k(3,2)}$. We have

$$J_1 = \{71, 64, P_1^3[(V_1, (0, 30, 30)), (V_2, (30, 22, 52)), (V_3, (52, 12, 64))]\}$$
$$J_2 = \{77, 30, P_2^2[(V_1, (0, 7, 37)), (V_2, (37, 23, 75))]\}$$

Clearly, J_1 and J_2 now are able to finish their execution at 64 and 75, respectively.

In this case, the system computation efficiency within $[0, 77]$ is calculated as: $1 - \frac{3}{64+30+3} = 97\%$, which is much higher compared with 51% and 91%. □

The system average processor utilization in Example1 is

$$\mathcal{A} = \frac{\frac{64}{71} + \frac{30}{77} + \frac{11}{72}}{3} = 0.48$$

From this example, we can make the following two conclusions: 1) if an online EDF job scheduling is employed and workload demand exceeds resources available, the online job removal is unavoidable; 2) different job removal policies can result in different system computation efficiency.

Another interesting observation is that in this example, the system's average processor utilization is only 48%, still job removal is unavoidable on V_2 in order to ensure the rest of jobs meet their end-to-end deadline. In next section, we present two computation efficiency driven job removal heuristics for meeting end-to-end deadlines in distributed real-time systems.

IV. Computation Efficiency Driven Job Removal Policies

Job removal and local deadline assignment procedures both need workload information on individual processors (V_x). Every time a new job arrives at processor V_x, the workload changes on V_x and so is the job execution priorities. Hence, new local deadlines are assigned to the job set including both the newly arrived jobs and jobs that are already in the queue for execution. The newly assigned local deadlines must satisfy the constraints given by (7) and (8). If no feasible local deadline assignment can be found, some jobs have to be removed from future execution so that the remaining jobs can meet their end-to-end deadlines. Our goal is to maximize the system computation efficiency for a given execution duration, i.e., maximize the ratio between computation contributed to jobs that are successfully completed before their end-to-end deadlines and the total computation the system has performed.

For a given period of time, the number of jobs released is fixed, hence, intuitively, the less the removed jobs, the higher the system computation efficiency. Unfortunately, due to the dynamic and distributed nature of the system, we are not able to predict how the current job removal decision may impact the number of future successful jobs and what these successful jobs are. We can only use local information to make heuristic decisions.

At a given time instance t, the information we have about a job released to processor V_x are: (1) job execution completion ratio $\eta(J_{i,j}, t)$, and (2) system potential computation efficiency if job $J(i, j)$ is removed, i.e., $\rho(J_{i,j}, t)$, where $J(i, j) \in \Omega(V_x)$ and $j = k(i, x)$.

In this section, we introduce two computation efficiency driven job removal policies: i.e., the Least Completion ratio First (LCF) and the Maximum Potential computation efficiency First (MPF) job removal policies.

A. The LCF Job Removal Policy

Before we present the job execution completion ratio based removal policy, we use an example to provide the intuition behind the policy.

Consider the Example 1 given in the previous section. At time 40, as the job set on V_2, i.e., $J_{1,k(1,2)}$, $J_{2,k(2,2)}$ and $J_{3,k(3,2)}$, does not satisfy schedulability constraints, one of the jobs has to be removed. Job J_1 has completed 40 time unit and has 24 time units of work left; job J_2 has completed 7 time units, with 23 time units of work left; while job J_3 has just arrived at V_2, and completed 3 time

units with 8 time units remaining for execution. Therefore, at time $t = 40$, the job execution completion ratios are $\eta(J_{1,k(1,2)}, 40) = \frac{40}{40+24} = 63\%$, $\eta(J_{2,k(2,2)}, 40) = \frac{7}{7+23} = 23\%$, and $\eta(J_{3,k(3,2)}, 40) = \frac{3}{3+8} = 27\%$. Based on the least job execution completion ratio first principle, job J_2 is to be removed, which is different from the job removal policy based on the unfinished execution time. Accordingly, all the jobs except J_2 can successfully complete their execution, and total $7 + 64 + 11$ time units of computation is performed by the system when $t = 77$. The system computation efficiency is thus $\mu(77) = \frac{75}{82} = 91\%$, which is higher than 51% attained by the maximum remaining execution time based policy.

There are situations where multiple jobs may share the same completion ratio. In this case, we use unfinished execution time to break the ties, i.e., we remove the job with the maximum remaining execution time to lessen the burden of the downstream processors. The details of the LCF policy is given in Algorithm 1. Assume the time point when the removal decision needs to be made is t and the job set that violates schedulability constraints is $\Omega(V_x)$, r is the job index chosen for removal.

Algorithm 1: LCF($\Omega(V_x)$,t)

```
1  Least_Compl_Ratio = 1
2  r = 0
3  foreach J_{i,k(i,x)} ∈ Ω(V_x) do
4      if η(J_{i,k(i,x)}, t) < Least_Compl_Ratio then
5          Least_Compl_Ratio = η(J_{i,k(i,x)}, t)
6          r = i
7      end
8      if η(J_{i,k(i,x)}, t) = Least_Compl_Ratio &&
       C_rem(J_{r,k(r,x)}, t) < C_rem(J_{i,k(i,x)}, t) then
9          r = i
10     end
11 end
12 Return r
```

In Algorithm 1, we compute each job's completion ratio in the job set and obtain the job(s) with the least job execution completion ratio among the set (line 4 to line 7). If there are multiple jobs sharing the same job execution completion ratio, we choose the one with the maximum remaining execution time (line 8 to line 10). Finally, we return the index of the chosen job (line 11). The time complexity of the algorithm is $O(n)$, where $n = |\Omega(V_x)|$.

The job execution completion ratio based removal policy focuses on the computation efficiency of individual jobs. Another aspect to consider for minimizing the computation efficiency is to focus on the potential computation efficiency on the local processor that can be brought by removing a job.

B. The MPF Job Removal Policy

Again consider Example 1 given in Section III. At time $t = 40$, as the job set on V_2, i.e., $J_{1,k(1,2)}$, $J_{2,k(2,2)}$ and $J_{3,k(3,2)}$, is not schedulable, one of them needs to be removed. If $J_{1,k(1,2)}$ is removed, we are assuming $J_{2,k(2,2)}$ and $J_{3,k(3,2)}$ are able to complete successfully, then $\rho(J_{1,k(1,2)}, 40) = \frac{30+11}{40+30+11}$. If $J_{2,k(2,2)}$ is removed instead, then $J_{1,k(1,2)}$ and $J_{3,k(3,2)}$ are assumed to complete successfully, $\rho(J_{2,k(2,2)}, 40) = \frac{64+11}{7+64+11}$. Similarly, $\rho(J_{3,k(3,2)}, 40) = \frac{64+30}{3+64+30}$. We thus remove $J_{3,k(3,2)}$ for the higher potential system computation efficiency. As we discussed in the previous section, the system computation efficiency following this policy is 97% at $t = 77$.

Algorithm 2 describes the MPF job removal policy. It is similar to Algorithm 1 except that rather than finding the jobs with the least completion ratio, it finds the job with maximum potential computation efficiency. The time complexity of this algorithm is the same as Algorithm 1.

Algorithm 2: MPF($\Omega(V_x)$,t)

```
1  Max_Potential_Effi = 0
2  r = 0
3  foreach J_{i,k(i,x)} ∈ Ω(V_x) do
4      if ρ(J_{i,k(i,x)}, t) > Max_Potential_Effi then
5          Max_Potential_Effi = ρ(J_{i,k(i,x)}, t)
6          r = i
7      end
8      if ρ(J_{i,k(i,x)}, t) = Max_Potential_Effi &&
       C_rem(J_{r,k(r,x)}, t) < C_rem(J_{i,k(i,x)}, t) then
9          r = i
10     end
11 end
12 Return r
```

C. Comparison of the Two Policies

Assume $\Omega(V_x)$ ($V_x \in \mathcal{P}$) is a job set that is not schedulable and $J_{r,k(r,x)}$ is the job chosen for removal at time point t_r, $0 \leq t_r \leq t$. Our goal is to maximize $\frac{E_s(t)}{E_f(t)+E_s(t)}$, i.e., minimize $\frac{E_f(t)}{E_f(t)+E_s(t)}$.

$$\min \frac{E_f(t)}{E_f(t) + E_s(t)} = \min \frac{\sum\limits_{J_r \in \Gamma_{\text{fail}} \wedge 0 \leq t_r \leq t} C_{\text{tot}}(J_{r,k(r,x)}, t_r)}{\sum\limits_{J_r \in \Gamma_{\text{fail}} \wedge 0 \leq t_r \leq t} C_{\text{tot}}(J_{r,k(r,x)}, t_r) + \sum\limits_{J_i \in \Gamma_{\text{succ}}} E_i} \tag{9}$$

The MPF policy makes job removal decisions based on system's potential computation efficiency and assumes that by removing one job from its further execution, all remaining jobs will complete successfully. In other words, the job selection for removal is based on (10):

$$\min_{J_{i,k(i,x)} \in \Omega(V_x)} \left\{ \frac{C_{\text{tot}}(J_{i,k(i,x)}, t)}{C_{\text{tot}}(J_{i,k(i,x)}, t) + \sum\limits_{J_{h,k(h,x)} \in \Omega(V_x) \wedge h \neq i} E_h} \right\} \tag{10}$$

When system's average processor utilization is low, it is more likely that removing one job is sufficient for the remaining jobs to complete successfully, and hence results in small failure job set Γ_{fail}. When $|\Gamma_{\text{fail}}|$ is small, comparing formula (9) to (10), it is highly possible that $C_{\text{tot}}(J_{i,k(i,x)}, t)$ and

$\sum_{J_r \in \Gamma_{\text{fail}}} C_{\text{tot}}(J_{r,k(r,x)}, t)$ are close to each other. If $|\Gamma_{\text{fail}}| = 1$, formula (9) and formula (10) have the same result. Therefore, when the average processor utilization is low, the MPF policy performs well in achieving the goal of maximizing system computation efficiency.

When the system average utilization is high, more jobs may miss their end-to-end deadlines, which indicates more computation may be wasted. In other words, for formula (9), $E_f(t)$ becomes the dominant component. In this case, keeping the waste ratio low is the key to reach the goal of high computation efficiency. The LCF policy removes the job with least job execution completion ratio, i.e.,

$$\min_{J_{i,k(i,x)} \in \Omega(V_x)} \left\{ \frac{C_{\text{tot}}(J_{i,k(i,x)}, t)}{Ei} \right\}$$

It hence performs better when system average utilization is high. Next section we empirically evaluate the proposed policies.

V. Performance Evaluation

In this section, we empirically evaluate the proposed computation efficiency driven job removal policies and compare them with an existing job removal policy based on job remaining execution times, i.e., the RET policy [8].

When job removal becomes necessary on a processor, the RET policy removes the job that has the maximum remaining execution time. As we do not know the future workload on downstream processors, removing the job with the maximum remaining execution time lessens the burden of downstream processors. Hence, the remaining jobs will have better chances to finish before their end-to-end deadlines. Furthermore, by removing the job that requires the longest future execution time, it is more likely to reduce the total number of removed job. From this point of view, the RET policy could indirectly lead to good system computation efficiency.

Our evaluation and comparison focus on two aspects, i.e., system computation efficiency and job removal ratio. For both of the evaluation metrics, we have to consider when the workload is balanced or unbalanced among all processors. The following gives the detail about our experiment settings, job set generations, and result analysis.

A. Simulator Implementation

The experiments are conducted on a simulator which we have developed. In the simulator, we use the OLDA algorithm [8] to assign job local deadlines. In the process of assigning local deadlines, if the constraints defined by (7) and (8) cannot be met by the job segments on a processor, a job removal algorithm is called to remove selected jobs from their further execution.

B. Job Set Model Generation

We use TGFF to randomly generate job models with execution path information. The nodes in TGFF graph have precedence constraints, which corresponds to the execution order of sub-jobs in our job model. Based on a series-parallel structure of the TGFF graph, we do the following mapping:

- A chain in a TGFF graph corresponds to an execution path of a job.
- A node in a TGFF graph corresponds to a job segment.
- An edge in a TGFF graph corresponds to a precedence constraint of the execution order between two sub-jobs.
- The number of chains corresponds to the number of jobs in the system.
- The height of a TGFF graph corresponds to the number of processors in the system.

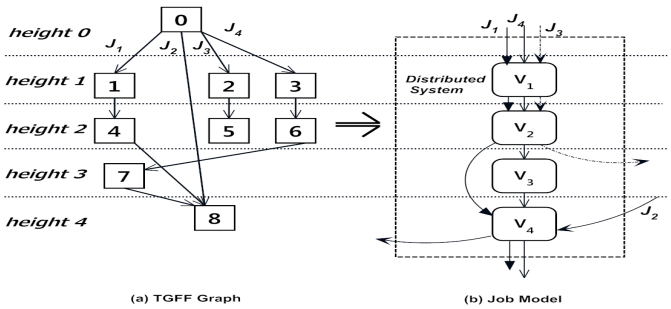


Fig. 2. Job Model Generation

Fig. 2(a) shows a task graph generated by TGFF. Based on the height from the root node (node 0), the graph has four (4) layers. Hence, we assume there are four processors. The job segments that are on the same layer are assigned to the same processor as shown in Fig. 2(b). More specifically, for the generated task graph shown in Fig. 2(a), we have the job model given below which is also graphically represented by Fig. 2(b):

$$J_1 = \{?, ?, P_1^3((V_1, (0, ?, ?)), (V_2, (?, ?, ?)), (V_4, (?, ?, ?)))\}$$
$$J_2 = \{?, ?, P_2^1((V_4, (0, ?, ?)))\}$$
$$J_3 = \{?, ?, P_3^2((V_1, (0, ?, ?)), (V_2, (?, ?, ?)))\}$$
$$J_4 = \{?, ?, P_4^4((V_1, (0, ?, ?)), (V_2, (?, ?, ?)), (V_3, (?, ?, ?)), (V_4, (?, ?, ?)))\}$$

It is worth point out that when we map a TGFF graph into a job model graph, we intentionally balance the numbers of sub-jobs going across different processors, i.e., the differences between the number of sub-jobs on each processor is less than 2. With the balanced number of sub-jobs on each processor, we can control the workload generation on processors which we will discuss in the following subsection.

Once the structure of the job model is generated, the next step is to generate timing information for each job, i.e., end-to-end execution time, end-to-end deadline, and execution time of each job segments, i.e., workload on each processors.

C. Workload Generation

As a job may go through a sequence of processors, hence the end-to-end execution time of a job is distributed among different sub-jobs. For a given end-to-end execution time of a job, how it is distributed along its execution path reflects the

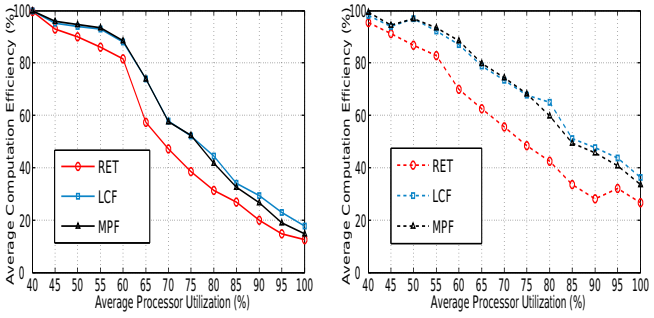
workload balance among different processors as in the task set model we generate, the number of sub-jobs on each processor are on purposely made roughly the same (Section V-B). We characterize two different workload scenarios: 1) balanced workload, i.e., the execution time of different job segments has a high probability to be the same; and 2) unbalanced workload, i.e., the execution time of different job segments has a high probability to be different. We use the methods presented in [14] to generate the two different types of workloads.

D. Experiment Setting

Jobs and their execution paths are randomly generated by TGFF. The number of jobs in a job set ranges from 15 to 30 and the number of processors in the system ranges from 5 to 10. For each job set model generated by the TGFF, we use the Uunifast method [14] to generate a set of job instances. For the generated set of job instances, we obtain system execution efficiency and job removal ratio based on the three different job removal policies, i.e., LCF, MPF, and RET policies. The experiment is repeated on 10 different job models, i.e., 10 different TGFF graphs, with 50 different sets of job instances. We take the average value of the $10 \times 50 = 500$ tests. The experiment results are plotted based on average processor utilization in the system ($\mathcal{A}$) which is defined in Definition 1.

E. Results Discussion

Under the experiment settings given above, the system computation efficiency under balanced workload and unbalanced workload are depicted in Fig. 3(a) and Fig. 3(b), respectively.

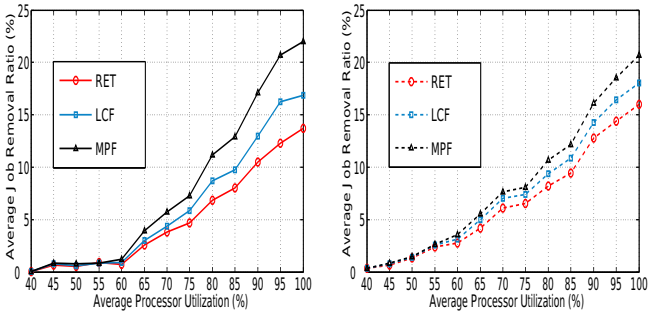


(a) Balanced Workload (b) Unbalanced Workload

Fig. 3. System Computation Efficiency

From Fig. 3, we have the following observations:

- System computation efficiency decreases as the average processor utilization increases. Furthermore, efficiency drops quickly when the average utilization is above 60% and 50% for balanced workload and unbalanced workload, respectively.
- System has higher computation efficiency under unbalance workload than under balanced workload. This phenomenon becomes more obvious when the average processor utilization is high. The reason is that unbalanced workload may result in early job removals especially when average processor utilization is high. When a job is removed at its early stage, less computation is wasted on the job and hence can result in higher computation efficiency.
- Under both balanced and unbalanced workloads, both LCF and MPF achieve higher computation efficiency than the RET policy. Furthermore, when the average processor utilization is low, MPF performs better than LCF. The position changes when the average processor utilization becomes high (above 75%). This observation is consistent with our previous analysis. However, the difference is not significant.
- The superiority of both LCF and MPF over RET with respect to system computation efficiency is more significant under unbalanced workload than under balanced workload. The difference can be as large as 20% between the LCF (MPF) and RET under unbalanced workload.



(a) Balanced Workload (b) Unbalanced Workload

Fig. 4. Job Removal Ratio

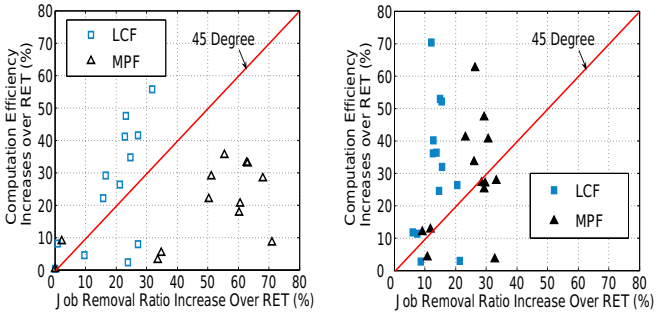
Fig. 4 depicts the average job removal ratio. Similar observations can be made as following:

- As average processor utilization increases, job removal ratio increases for all three approaches.
- Under balanced workload, when the average processor utilization is above 60%, the job removal ratio increase quickly. For unbalanced workload, the increasing of the job removal ratio is relatively smooth compared to its balanced counter part.
- Both LCF and MPF job removal policies result in higher job removal ratio than the RET policy. The difference on average is smaller under unbalanced workload (0.9% and 1.8% for LCF and MPF, respectively) than under balanced workload (1.2% and 3% for LCF and MPF, respectively).
- The LCF policy has less job removal ratio than MPF policy under both balanced and unbalanced workloads.

Both LCF and MPF focus on maximizing the system computation efficiency while the RET focus on minimizing job removal ratio, hence, as can be seen from Fig. 3, the computation efficiency differences between LCF and MPF is small compared with the differences between LCF (MPF) and RET, but both LCF and MPF have better computation efficiency than the RET policy does. However, from Fig. 4, RET outperforms LCF and MPF in terms of average job

removal ratio. In other words, both LCF and MPF obtain better system computation efficiency at the cost of increased average job removal ratio. On the other hand, RET has low average job removal ratio but at the cost of decreased system computation efficiency. Fig. 5 shows the trade-offs between computation efficiency and job removal ratio which is obtained by integrating Fig. 3 and Fig. 4 as following:

For each average processor utilization, we obtain from Fig. 3 and Fig. 4 the computation efficiencies, i.e., e_L, e_M, and e_R, and job removal ratios, i.e., r_L, r_M, and r_R, for the LCF, MPF, RET, respectively. We calculate both the computation efficiency and job removal ratio percentage increases of LCF and MPF over RET. The value pairs $(\frac{r_L - r_R}{r_R}, \frac{e_L - e_R}{e_R})$ (and $(\frac{r_M - r_R}{r_R}, \frac{e_M - e_R}{e_R})$) are plotted on Fig. 5.



(a) Balanced Workload (b) Unbalanced Workload

Fig. 5. Tradeoff Between LCF (MPF) and RET

In Fig. 5, a data point residing above the 45 degree line represents the benefit in terms of increased computation efficiency of LCF (MPF) over RET; while a data point residing below the 45 degree line represents the increased job removal ratio.

From Fig. 5, we have the following observations:

- Under balanced workload, LCF outweighs RET as most of the data points are above the 45 degree line. On the other hand, the computation efficiency improvement brought by MPF is at a high cost of job removal ratio.
- Under unbalanced workload, both LCF and MPF outweigh RET when both computation efficiency and job removal ratio are taken into consideration.
- MPF performs better under unbalanced workload than under balanced workload.

From these experiments, we can conclude that LCF performs the best when the system average processor utilization is high, and MPF has advantage over LCF when the average processor utilization is relative low. Both the proposed approaches outperforms the RET approach even when job removal ratio is taken into consideration.

VI. Conclusion

As the amount of energy consumed by computer systems increases dramatically each year, how to most efficiently utilize the computer resources and increase their computation efficiency has become an important and practical issue. In this paper, we present two computation efficiency driven job removal policies, i.e., LCF and MPF policies, for meeting end-to-end deadlines in distributed real-time systems. To our best knowledge, we are the first to consider system computation efficiency for online job scheduling. The performances with respect to system computation efficiency and job removal ratio of the two polices are fully investigated under different job model, different average processor utilization, and balanced and unbalanced workload distributions. The experiment results clearly show the advantages of the proposed policies. However, in this work, we assume that each processor consumes the same amount of energy for a unit of work, hence computation efficiency can be directly mapped to energy consumption efficiency. Our immediate future work is to study the online job removal policy when processors in a distributed system have different energy consumption rates.

References

[1] R. Dick, D. Rhodes, and W. Wolf, "Tgff: task graphs for free," in *Proceedings of the 6th international workshop on Hardware/software codesign*. IEEE Computer Society, 1998, pp. 97–101.

[2] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Period optimization for hard real-time distributed automotive systems," in *Proceedings of the 44th annual Design Automation Conference*. ACM, 2007, pp. 278–283.

[3] S. Matic and T. Henzinger, "Trading end-to-end latency for composability," in *Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International*. IEEE, 2005, pp. 12–pp.

[4] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*. IEEE, 2012, pp. 145–154.

[5] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*. IEEE, 2010, pp. 13–22.

[6] J. Peha, "Heterogeneous-criteria scheduling: minimizing weighted number of tardy jobs and weighted completion time," *Computers & operations research*, vol. 22, no. 10, pp. 1089–1100, 1995.

[7] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, "Dynamic thermal management through task scheduling," in *Performance Analysis of Systems and software, 2008. ISPASS 2008. IEEE International Symposium on*. IEEE, 2008, pp. 191–201.

[8] S. Hong, T. Chantem, and X. Hu, "Meeting end-to-end deadlines through distributed local deadline assignments," in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*. IEEE, 2011, pp. 183–192.

[9] J. Koomey, "Estimating total power consumption by servers in the us and the world," 2007.

[10] M. Webb *et al.*, "Smart 2020: Enabling the low carbon economy in the information age," *The Climate Group. London*, vol. 1, no. 1, pp. 1–1, 2008.

[11] N. Hardavellas, "Exploiting dark silicon for energy efficiency," *NSF Workshop on Energy-Efficient Data Management*, 2011.

[12] H. Chetto, M. Silly, and T. Bouchentouf, "Dynamic scheduling of real-time tasks under precedence constraints," *Real-Time Systems*, vol. 2, no. 3, pp. 181–194, 1990.

[13] H. Chetto and M. Chetto, "Scheduling periodic and sporadic tasks in a real-time system," *Information Processing Letters*, vol. 30, no. 4, pp. 177–184, 1989.

[14] E. Bini and G. Buttazzo, "Biasing effects in schedulability measures," in *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*. IEEE, 2004, pp. 196–203.