Real-Time Scheduling in Cyber-Physical Systems

Yanwen Chen*. Yixiang Chen.**

**Software Engineering Institute of East China Normal University, ShangHai, China*
(Tel: 021-62235380; e-mail: ywchen@gmail.com).
*** Software Engineering Institute of East China Normal University, ShangHai, China*
(e-mail: yxchen@sei.ecnu.edu.cn)

Abstract: Cyber-Physical System is a new frontier appearing with many challenges. Real-time issue is an important one in those challenges. To address the problem, in this paper, real-time scheduling algorithms tailored for CPSs are proposed. They are designed for the purpose of maximizing system utilization, reducing scheduling overhead and decreasing context switch cost. Simulation results demonstrate that the strategy of splitting task can improve the system utilization. However, after the system utilization surpasses 75%, the success ratio will reduce because large numbers of splitting tasks have an adverse impact on it.

Keywords: cyber physical system, distributed system, real-time scheduling.

1. INTRODUCTION

Cyber-Physical Systems (CPSs) are distributed and hybrid real-time dynamic systems with complex communication, providing real-time monitoring and actuation services (Rajkumar, 2007). They are a paradigm of enabling ubiquitous computing to everything including physical processes and objects at large-scale (Sha et al., 2008). From the view of Lee (2006), CPSs are a comprehensive concept systems which consist of a great number of distributed computing devices tightly coupled with their physical environments. Sensing layer, transport layer and information processing layer are three basic layers for CPSs. Physical information are detected by sensors in sensing layer. Then, the information is sent to processing layer through transport layer. At last, processing layer processes the information and then sends appropriate decisions to physical actuators. From this perspective, many existing network such as satellite network, mobile network, wireless sensor network, embedded system, even the internet are all fall into the concept of CPSs (Koubaa and Andersson, 2009). However, CPSs are not a simple combination of these systems. New technologies need to be devised to overcome challenges (Lee, 2008), such as real-time issue, data aggregation, robustness, safety and security (Mueller, 2006).

One scenario of real-time issue is intelligent transportation system (ITS) in which timely dissemination of traffic-related information to drives is a key property. Car drivers often experience getting stuck in a traffic jam on their way to work even they have checked the traffic reports before leaving their homes. These unanticipated events can have an adverse impact. For example, business professionals may miss important meetings or flights. Sometimes even worse, doctors or fireman may be blocked when they are on the way to save people's life.

Supporting the application requires a wireless network formed among the vehicles and an advanced information processing platform used to process data from sensors (Rajkumar et al., 2010). CPSs provide a natural framework to address these challenges since they integrate the physical, cyber and human factors in framework. In addition, massive information from thousands of vehicles needs to be processed in the processing platform. So it's hard to imagine that without scheduling analysis in the platform, the real-time performance can be guaranteed.

So in the paper, scheduling algorithms, as one solution, are proposed for the real-time issue. Real-time scheduling problem in embedded system has been studied for more than a decade, producing fruitful research results. Multiple-core and real-time scheduling (Mollison et al., 2010) in embedded system can be a good foundation for us to design a scheduling algorithm for CPSs. However, CPSs go beyond traditional embedded systems. Traditional embedded systems are closed, not only in the sense of closed physical locations or dedicated networks, but also closed with respect to their computational boundaries, i.e., all the participation elements in the systems are known initially (Talcott, 2008). By comparison, CPSs are shifting towards openness and wide-scale which lead to a novel scheduling analysis on real-time issue. In the other hand, CPSs require distributed architectures to support safety critical real-time control (Kang and Son, 2008). Traditional distributed system emphasis on the paralleling processing which can reduce the computing time of tasks but can not guarantee a real-time response. So novel scheduling algorithms which meet the requirement of real-time and system environments of CPSs are explored in the paper.

This paper makes the following contributions to CPS-based solutions for real-time issue: First, a heuristic task assignment

algorithm is explored to minimize the scheduling overload when a coming task is assigned to a cluster. Not liking the traditional assignment strategy which balances the numbers of task, the new strategy balances the scheduling overload. After assignment, in order to maximize the utilization of processors, a task splitting algorithm is devised when a task cannot be accommodated as a whole by any cluster. Not liking distributed computing in which each split tasks are executed parallel, in the paper, these split tasks which is split from the same original task have to be executed serially even they are assigned into different clusters. So the novel task splitting algorithm tries to minimize the splitting granularity when maximize the utilization of clusters. At last, a global scheduling algorithm which follows the idea of BFair algorithm proposed by Zhu (2003) is designed. But not liking the BFair algorithm which focuses on the fairness of scheduling, the new global scheduling algorithm pays more attention to the real-time performance and system utilization.

The novel points of the paper are: first, distributed clusters are used to process information, because the scalability of distributed clusters have capability to process huge information from sensing layer, which guarantee the real-time performance for CPSs; second, the proposed scheduling algorithms are tailored for CPSs environments since the network delay between clusters are taken into account; third, strategies of maximizing system utilization, reducing scheduling overhead and context switch cost are considered. All these points contribute to the real-time performance of CPSs.

The remainder of this paper is organized as follows. Section 2 describes some prior research works. Section 3 presents an abstract system model and describes existing issues. Section 4 demonstrates three algorithms used to address these issues. At last, section 5 shows the simulation results and section 6 gives some conclusions.

2. RELATED WORK

Prior research closely related to the real-time issue of CPS can be classified along two dimensions:

(1) Those that designing a time-related models or architectures for CPSs to meet its real-time requirements. Tan et al. (2010) introduced a concept lattice-based event model for CPSs. Under the model, a CPSs event is uniformly represented by three components: event type, its internal attributes and its external attributes. The event model provides several advantages in terms of flexibility, time-related QoS requirement and complexity. Benveniste (2010) proposed a loosely Time-Triggered Architectures for CPSs, in which computation and communication units are all triggered by autonomous, non synchronized clocks. So communication media act as shared memories between writers and readers and communication is not blocking. Zhang et al., (2009) introduced a dynamic battery model which supports a co-design approach for CPSs. Their model allows online prediction of battery life, which benefit for optimal discharge profile. Sha and Meseguer (2008) proposed a novel paradigm for CPS to embody design rules of complexity-control nature in highly reusable, very robust and formally verified architectural patterns.

(2) Those that adding some mechanisms or middleware to overcome the real-time challenge of CPSs. Zimmer et al. (2010) presents three mechanisms for time-based intrusion detection. The mechanism can detect the execution of unauthorized instructions in real-time CPS environment. Huang et al. (2010) presents a case study of several fundamental interlocking challenges in developing CPS for real-time hybrid structural testing. Tidwell et al. (2010) designed an optimal utility accrual scheduling policy for CPS with periodic, non-preemptable tasks. They devised the policies by solving a Markov Decision Process formulation of the scheduling problem. Zhang and Gill (2008) proposed an effective configurable component middleware approach in supporting different applications with aperiodic and periodic events and providing a flexible software platform for distributed cyber-physical systems. Gokhale and McDonale (2010) proposed a combination solution of cyber (e.g., Behaviour of protocols such as TCP, IP and 802.11 MAC) and physical (e.g., vehicular speed, radio range) for Intelligent Transportation System. Zhang and Shi (2008) study the task scheduling problem of CPS from the aspect of its feedback control behaviors. They co-design the control law and the task scheduling algorithm for predictable performance and power consumption for both computing and the physical system.

These prior efforts are important since they provide crucial insights into the time-related model and communication behaviours of CPSs, or contribute to mechanisms and strategies that can tune real-time performance.

3. INFORMATION PROCESSING MODEL AND PROBLEM DESCRIPTION

The paper focuses on scheduling problems in the information processing layer of CPSs. The layer is abstracted with one master node and several distributed clusters. The master node takes charge of receiving and assigning tasks. Clusters are responsible for executing tasks.

3.1 Information Processing Model

The information processing model consists of one master node and m clusters $\boldsymbol{Cl} = \{cl_1, cl_2, ..., cl_m\}$. Clusters are connected each other by internet. For simplification, it is assumed that each cluster has k processors as Fig.1 shows. The communication cost, task migration cost and context switch cost in a cluster are ignored. But the communication cost between clusters (assume it is a constant C_0) is taken into account. The utilization of task set is assumed to be U and the formula $U \le k \cdot m$ should be met. (Otherwise, task set is not schedulable).

3.2 Task Model

For a task set $\boldsymbol{T} = \{T_1, T_2, ..., T_n\}$, assume that preemption is allowed and there is no dependency between tasks. For

simplification, just periodical tasks are discussed in the paper. Each periodical task T_i= (c_i, p_i, r_i, d_i) is defined with four parameters, where c_i is the task's worst-case execution time, p_i is its period, r_i is its release time and d_i is its deadline. Assume that c_i, p_i, r_i, and d_i are integers, denoting the number of time quanta. Moreover, it is assumed that tasks are synchronous. The utilization of task T_i is defined as $u_i=c_i/p_i$ and the system utilization is defined as the summation of all task's utilization $U=\sum u_i$ ($1\le i\le n$). Assume that the different parts of a split task cannot be executed in parallel. For each task there is $u_i\le 1$. If a task is not split, then $r_i=0$ and $p_i=d_i$, otherwise the r_i and p_i will be reset (details will be discussed in section 4.2).

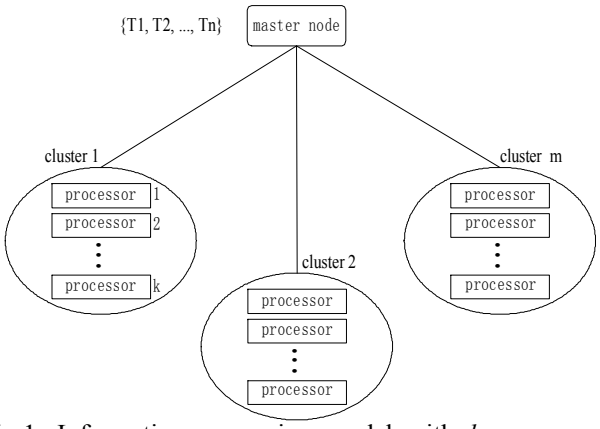

Fig.1. Information processing model with k processors in each cluster

3.3 Problem Description

In the intelligent transportation system (ITS), for example, some car drivers want to know about the traffic information along the path to their destination, some drivers need to know if there is car coming from other side and how much speed the car is, especially in the cross road. All these requirements can be done by sending them to the information processing layer of ITS. The system can use powerful clusters to process massive requirement. At first, the master node receives these requirements/tasks and then assigns them to clusters. How to find an optimal cluster to minimize the scheduling overhead is the first problem should be solved. When no more cluster can be used to accommodate a task, the master node start to split the task and assign each of these splitting parts to clusters. How to split a task to minimize the splitting granularity is the second problem that cannot be ignored. At last each cluster start to execute these tasks assigned to them. How to design an optimal real-time global scheduling algorithm is the third problem need to be addressed. To solve these problems, three algorithms have been proposed in the paper.

4. ALGORITHMS

In the paper, scheduling points are defined as period boundaries between 0 and $Lcm(\boldsymbol{T})$, in which $Lcm(\boldsymbol{T})$ means the least common multiple of a set of periods ($\{p_1, p_2, \ldots p_n\}$) of the task set $\boldsymbol{T}$. For example, if a task set includes 3 tasks ($\boldsymbol{T}=\{T_1, T_2, T_3\}$) with period p_1=5, p_2=6, and p_3=15. So its least common multiple number $Lcm(\boldsymbol{T})$ is 30 and the scheduling points are 5, 6, 10, 12, 15, 18, 20, 24, 25, and 30.

4.1 Task Assignment

The objective of the task assignment algorithm is to assign n tasks to m clusters such that the total scheduling points are minimal. This is a NP-hard problem since it can be described as a bin-packing problem as below:

Problem Statement: Given m nodes with k processors for each. For n period tasks $\{T_1, T_2, \ldots, T_n\}$ with utility $u_i\le 1$ ($1\le i\le n$), find if there is an integer B ($B\le m$) and B-partition $S_1 \cup S_2 \ldots \cup S_B$ of $\{1,\ldots,n\}$ such that minimize $\sum sp_j$ (sp_j means the number of scheduling point at node j, $j\in[1,B]$), in the constraint of $\sum u_i\le k$ ($i\in S_w$) for all w=1,…,B.

For this NP-hard problem, one approximate solution is greedy algorithm. In the paper, "*contribute factor*" is designed to evaluate the increase of scheduling points after adding a new task. So the "*contributed factor*" needs to be calculated once the new task is assigned into a cluster. The cluster which has the smallest contribute factor will be chosen to accept the new task. The "*contribute factor*" can be calculated using inclusion-exclusion principle. Suppose the period of a new task is A. For the m^{th} cluster cl_m which includes a task set $\boldsymbol{T}=\{T_1, T_2, \ldots, T_n\}$, before adding a new task, the number of scheduling points is

$$SP(cl_m) = l(cl_m)\Big[\sum_{1<i\le n}(1/p_i) - \sum_{0<i<j\le n}(1/l(p_i,p_j)) + \sum_{0<i<j<k\le n}(1/l(p_i,p_j,p_k)) - \ldots + (-1)^n (1/l(p_1,p_2,\ldots p_n))\Big] \quad (1)$$

After adding a new task with period A, the number of scheduling points is

$$SP(cl_m, A) = l(cl_m, A)\Big[\sum_{1<i\le n+1}(1/p_i) - \sum_{0<i<j\le n+1} 1/l(p_i,p_j) + \sum_{0<i<j<k\le n+1} 1/l(p_i,p_j,p_k) - \ldots + (-1)^{n+1}(1/l(p_1,p_2,\ldots p_n,A))\Big] \quad (2)$$

The explanations of notations are as below:

- $SP(cl_m)$ means the number of scheduling points of the m^{th} cluster.
- $SP(cl_m, A)$ means the number of scheduling points of the m^{th} cluster after accepting the new task which period is A.
- $l(p_i,p_j)$ means the least common multiple of number p_i and p_j.
- $l(cl_m)$ means the least common multiple of the period value of task set in cluster m.
- $g(l(cl_m),A)$ is the great common divisor of $l(cl_m)$ and A.
- $cop(clm, A)= g(l(clm),A)/A$, which demonstrates the coupling of the number A with the period value of task set in the cluster cl_m.

Since the formula $l(cl_m,A)=l(cl_m)\cdot A/g(l(cl_m),A)$ is true, so the "*contribute factor*" (the increase of the number of scheduling points) ΔSP is

$$\Delta SP = SP(cl_m, A) - SP(cl_m) \tag{3}$$
$$= l(cl_m)[a(cl_m) / cop(cl_m, A) + b(cl_m, A) / cop(cl_m, A) + a(cl_m)]$$

in which

$$a(cl_m) = \sum_{i=1}^{n} (1/p_i) - \sum_{0<i<j\le n} (1/l(p_i, p_j)) \tag{4}$$
$$+ \sum_{0<i<j<k\le n} (1/l(p_i, p_j, p_k)) - \ldots + (-1)^n (1/l(p_1, p_2, \ldots p_n))$$

$$b(cl_m, A) = 1 / A - \sum_{0<i\le n} (1/l(p_i, A)) \tag{5}$$
$$+ \sum_{0<i<j\le n} (1/l(p_i, p_j, A)) - \ldots + (-1)^{n+1} (1/l(p_1, p_2, \ldots p_n, A))$$

The ΔSP demonstrates that the more coupling between A and period values of task set, the fewer increasing of the scheduling points the cluster has. So a heuristic algorithm is proposed as shown in algorithm 1. The objective of this algorithm is to choose an appropriate cluster for a new arriving task such that the total increase of schedule points is minimal. Obviously it's not the optimal algorithm, because the ΔSP is also related to the original task set assigned to the cluster. But it does decrease the time complex:

Algorithm 1.

Assumptions:
1. There are m clusters with k processors in each.
2. Task set $\boldsymbol{T} = \{T_1, T_2, \ldots, T_n\}$ has been assigned into a cluster cl_i and it has been sorted by its period($p_1<p_2<\ldots<p_n$).
3. A new coming task T_{new} (c_{new}, p_{new}, r_{new}, d_{new}). So the utility of the new task is $u_{new}=c_{new}/(d_{new}-r_{new})$

1. $l(cl_i)=0$, $u(cl_i)=0$ $(i=1,2\ldots m)$
2. **For** each cl_i **do**
3. **If** $(u(cl_i)+u_{new}\le k)$ **then**
4. $cop_i = g(l(cl_i), p_{new})/ p_{new}$;
5. **else**
6. $cop_i = -\infty$;
7. **End For**
8. $cop_{max} = max(cop_1, cop_2, \ldots, cop_m)$;
9. set $S= \emptyset$;
10. put T_i into set S when $cop_i=cop_{max}$;
11. find the cluster cl_j such that the $l(cl_j)$ is minimal ($cl_j \in S$);
12. $cl_i = cl_i \cup T_{new}$;
13. $l(cl_j) = l(l(cl_j), p_{new})$;
14. $u(cl_i) \mathrel{+}= u_{new}$;

Some notations in the algorithm are explained as below:

- cl_i: the i^{th} cluster.
- $l(cl_i)$: the least common multiple of period value of task set in cl_i.
- $u(cl_i)$: the utility of cl_i.

The algorithm firstly checks if the cluster can accommodate the new task. If yes, it calculates the coupling value between the new task and the tasks already in the cluster (line4). Then set S collects the tasks which have the largest cop value (line10). And then find the cluster which has the smallest value $l(cl_j)$ with the task choosing from set S and put the task into the cluster (line12). At last, update the utility and the least common multiple number of the cluster (line13, 14).

The complex of the algorithm is $O(m)$, in which m is the number of clusters. Although it is not an optimal algorithm, the time complex can be reduced to polynomial complex.

4.2 Task Splitting

After assigning some tasks, if there is no one cluster can afford enough capacity for a new coming task, the strategy of task splitting is considered in order to increase the utilization of clusters. The strategy is taken only if the condition of $c_i/p_i \le \sum Ur_j$ $(1\le i\le n;\ 1\le j\le m)$ and $(qc_0+c_i)\le p_i$ is met. Ur_j is the rest utilization of cluster j; q is the number of clusters chosen for loading the task and c_0 is the communication cost between nodes. The communication time of context switch and task migration in a cluster is ignored. But the communication cost between clusters cannot be ignored. Note in order to decrease the communication overhead between clusters, after a task is assigned into a cluster, it does not be migrated to other clusters.

The aim of this part is try to execute more tasks by making full use of processors of clusters. But in this way, the scheduling points will be increased and inevitably the communication cost between clusters is raised. So a task do not be split if there still at least one cluster can accommodate it. Besides, in order to reduce the granularity of splitting, clusters are sorted by Ur with decreasing order. The cluster with largest Ur will be the highest priority to be chosen for allocating split tasks. By this way, the granularity of splitting can be as less as possible. Algorithm 2 shows the splitting algorithm in which the release time and deadline of each part of split task are set.

Some notations in the algorithm are explained as below:

- S_u: the sum of utilities which have been assigned to the new task T_{new}.
- *timeCost*: the total time taken for the new task, including computation time and context switch time between cluster.
- Ur_i: the rest utility of the i^{th} cluster.
- c_i : the new task's computation time in the i^{th} cluster.
- c_t: the total computation time of the new task in all clusters.
- U_{new}: the utility of the new task.
- $deadline_j$: the deadline of the partial task which has been assigned to the j^{th} cluster.

- *releaseTime*$_j$: the release time of the partial task which has been assigned to the j^{th} cluster.

Algorithm 2.

Assumptions:
1. A new coming task T_{new} (c_{new}, p_{new}, r_{new}, d_{new}). So the utility of the new task is $U_{new}=c_{new}/(d_{new}-r_{new})$
2. m clusters is sorted by its rest utility($Ur_1>Ur_2>\ldots>Ur_m$)
3. The communication cost between clusters is c_0.

1. $S_u=0$; $i=0$; $j=0$;
   relaseTime = 0; deadline=0, timeCost = 0, $c_t=0$;
2. **While**($S_u < U_{new}$ && *timeCost*$< p_{new}$ && $i<m$) **do**
3. S_u += Ur_i;
4. *timeCost* += $\lceil U r_i \cdot P_{new} + C_0 \rceil$;
5. $c_i = Ur_i * p_{new}$;
6. c_t += c_i;
7. i++;
8. **End While**
9. **If** ($S_u = U_{new}$) **then**
10. **While**($j<i$) **do**
11. $deadline_j = releaseTime_j + c_j * p_{new}/c_t - c_0/2$;
12. $releaseTime_{j+1} = deadline_i + c_0$;
13. j++;
14. **End While**
15. **End If**

In the first while loop, if a task still need more capacities, and at the same time, there exists a cluster which can provide capacity for it (line2), the cluster is chosen to allocate the rest of the task(line3, 4, 5, 6). In the second while loop the available time (*deadline* minus *releaseTime*) is reset for the split task according to the proportion of the computation time (c_i) taken from the total execution time (c_t) (line11, 12).

The complex of the algorithm is *O(m)*, in which m is the number of clusters.

4.3 Global Scheduling

The algorithm proposed in this part follows the idea of BFair global scheduling algorithm proposed by Zhu et al. (2003). The main difference between them is that the modified algorithm has capability of handling task set which includes split tasks while BFair focuses on the fairness of scheduling. For simplification, the situation with just one split task in a cluster is considered.

Before presenting the algorithm, some definitions are given. When the section $[b_k, b_{k+1}]$ is allocated, the mandatory unit is defined as $m_i^{k+1}=\max\{0,\lfloor RW_i^k+(b_{k+1}-b_k)w_i \rfloor\}$, in which RW_i^k is allocation error. The allocation error for task T_i at boundary time b_k is defined as the difference between $b_k \cdot w_i$ and the time units allocated to T_i before b_k. The m_i^{k+1} is the integer part of the summation of the remaining work in the interval of $[b_{k-1}, b_k]$ and the work to be done during $[b_k, b_{k+1}]$. The pending work is defined as $PW_i^{k+1}=RW_i^k+(b_{k+1}-b_k)w_i-m_i^{k+1}$ which is the corresponding decimal part. If T_i gets one optional unit when allocating $[b_k, b_{k+1}]$, then $o_i^{k+1}=1$, otherwise $o_i^{k+1}=0$. After allocating resources in the interval of $[b_k, b_{k+1}]$, the equation $RW_i^{k+1}=PW_i^{k+1}-o_i^{k+1}$ can be gotten. At boundary time b_{k+1}, the task T_i is ahead if $RW_i^k<0$, punctual if $RW_i^k=0$ and behind if $RW_i^k>0$.

The algorithm is presented in Algorithm 3, where *EX* is the extra task units which cannot be assigned to resource. It used to determine how many mandatory units should be removed. *RU* is the remaining units after allocating tasks' mandatory units. It used to determine how many optional units need to be allocated. Initially, $RW_i^0=0$ *(i = 1, 2,...).*

Algorithm 3.

Assumptions:
1. Task set $\boldsymbol{T}=\{T_1, T_2, \ldots, T_n\}$ in a cluster.

1. **For** ($T_1, T_2, \ldots, T_n$) **do**
2. $m_i^{k+1}=\max\{0,\lfloor RW_i^k+(b_{k+1}-b_k)w_i \rfloor\}$;
3. **End For**
4. **If** ($\Sigma m_i^{k+1}>(b_{k+1}-b_k)m$) **then**
5. $EX=\Sigma m_i^{k+1}-(b_{k+1}-b_k)m$;
6. *SelectedTaskSet=PickLowestPriorityTask*($EX, m_i^{k+1}=1, T_i \neq T_s$)
7. **For** ($T_i \in$ *SelectedTaskSet*) **do**
8. m_i^{k+1}--;
9. **End For**
10. **else**
11. $RU=m(b_{k+1}-b_k)-\Sigma m_i^{k+1}$;
12. *SelectedTaskSet = PickHighest PriorityTask*(*RU*)
13. **For** ($T_i \in$ *SelectedTaskSet*) **do**
14. $o_i^{k+1}=1$;
15. **End For**
16. **End If**
17. **For**($T_1, T_2, \ldots, T_n$) **do**
18. $PW_i^{k+1}=RW_i^k+(b_{k+1}-b_k)w_i-m_i^{k+1}$
19. $RW_i^{k+1}=PW_i^{k+1}-o_i^{k+1}$
20. **End For**

21. *GenerateSchedule(b_k, b_{k+1})*;

In the first "For" loop, the algorithm allocates mandatory units for each task T_i according their weights (line 2). If the section [b_k, b_{k+1}] has no enough resource to allocate these mandatory unit (*EX>0*), then the function of "*PickLowestPriorityTask*" will return the *EX* lowest priority tasks (line6) and each of them will reduce mandatory unit by 1 (line8). If there are time units left (*RU>0*), the function of "*PickHighestPriorityTask*" will return the *RU* highest priority task (line12) and each of them will get one optional unit (line14). After allocating all time units, the algorithm will calculate *PW* and *RW* for each task (line 18, 19). At last, the schedule for section [b_k, b_{k+1}] is generated by function *GenerateSchedule()*, which sequentially packs tasks to resources (line21).

Since a split task T_s (c_s, p_s, r_s, d_s) is a part of task T_i (c_i, p_i, r_i, d_i), the formula $p_s = p_i$, $c_s < c_i$, $d_s - r_s \subseteq d_i - r_i$ should be met. For a normal task, its weight is defined as $W_i = c_i/p_i$. For a split task T_s the weight is defined as

$$W_S=\begin{cases} c_S/(d_S-r_S) & [b_k,b_{k+1}]\in(np_S+r_S,np_S+d_S),n=0,1,... \\ 0 & others \end{cases} \quad (6)$$

The task's priority is sorted according to a characteristic string which follows the idea of Sanjoy et al. (1996). Define $\alpha_k(T_i)=sign[b_k\cdot w_i-\lfloor b_{k-1}\cdot w_i\rfloor-(b_k-b_{k-1})]$, which presents whether the task T_i can get enough resource or not in the section [b_k, b_{k+1}]. If yes ($\alpha_k(T_i)<0$), the task will be ahead at boundary b_{k+1}. If $\alpha_k(T_i)=0$, the task will be punctual at that boundary, otherwise, the task will be late. The urgent factor is defined as $UF_i^k=(1-(b_k\cdot w_i-\lfloor b_k\cdot w_i\rfloor))/w_i$, which is the minimal time needed for a task to collect enough work demand to receive one unit allocation and become punctual after b_k. After using the compare (T_i, T_j) algorithm proposed by Zhu et al. (2003), we can get the task set sorted by its priority. Then the function "*PickLowestPriorityTask*" is used to pick *EX* lowest priority tasks which mandatory unit is more than 1, excluding the split task. The "*PickHighestPriorityTask*" is used to choose *RU* tasks with highest priority from the sorted task set.

Theoretically, without any cost consuming, the algorithm can guarantee that each task can meet its deadline. This can be gotten by method of reduction to absurdity. If the algorithm cannot guarantee the real-time property, it means that there is at least one task which fails to meet its deadline. But from the algorithm we can see that once there is an optional unit, one task must be assigned unless there are no more tasks to be executed. In another word, processors are always busy to execute tasks. So if there is one task fails to meet it deadline, then, for the cluster which includes k processors, the total utilization of the task set assigned to the cluster must greater than k, which is contradict with the condition $u(cl_i)\leqslant k$. But practically, some tasks maybe miss their deadline because of the preemption cost, context switch cost, communication cost and so on.

4.4 Sample execution of the algorithm 3

In order to demonstrate the difference between algorithm 3 and BFair algorithm, the task set is chosen as below: T_1=(2,5,0,5),T_2=(3,15,0,15),T_3=(3,15,0,5),T_4=(2,6,0,6),T_5=(20,30,0,30),T_6=(6,30,0,30). Here T_3 is a split task, which means that the task T_3 can only be executed in [$z\cdot p_3$+0, z $\cdot$ p_3+5] (z is an integer). Assume these tasks will be executed in a cluster which includes two processors. The execution of Algorithm 3 is illustrated in the Table 1 of Appendix A.

Initially $RW_i^0=0$ (i=1…6). For the first section [b_0, b_1], $\alpha_k(T_i)$ and UF_i^k are calculated firstly for each task and then their results are put into the column b_1. For example, $\alpha_1(T_1)=sign[b_1\cdot w_1-\lfloor b_0 w_1\rfloor-(b_1-b_0)]=sign[5\cdot 2/5-\lfloor 0{\cdot}2/5\rfloor-(5-0)]=$ '–'

$UF_1^1=(1-(b_1\cdot w_1-\lfloor b_1\cdot w_1\rfloor))/w_1=(1-(5\cdot 2/5-\lfloor 5{\cdot}2/5\rfloor))/(2/5)=5/2$.

Then using the formula $m_i^{k+1}=\max\{0,\lfloor RW_i^k+(b_{k+1}-b_k)w_i\rfloor\}$ to calculate the mandatory units for each task, for example, T_1's mandatory unit is $m_1^1=\max\{0,\lfloor 0+(5-0)\cdot 2/5\rfloor\}=2$. The mandatory units for other tasks can also be calculated and the results are $m_2^1=1; m_3^1=3; m_4^1=1; m_5^1=3; m_6^1=1$ separately. Since $\Sigma\, m_i^1=11$ ($1\leq i\leq 6$) and the total available time units are $(b_1-b_0)\cdot m=(5-0)\cdot 2=10$. There is one extra unit (*EX*=1) need to be removed. Notices that T_5 has the lowest priority with $UF_5^1=5$, so one mandatory unit is removed from it. At this moment the allocation for the section [0, 5) is complete and then their corresponding *PW* and *RW* is calculated. All the values calculated above are filled into the column of b_1. The same procedure is repeated when calculating other sections.

For section [5, 6), the split task T_3 cannot get any resource since the section exceeds its deadline. So in this section, just tasks T_1, T_2, T_4, T_5, and T_6 calculate their mandatory unit. As a result, T_4 and T_5 get one unit separately. And then each task except the split task T_3 calculates their *PW* and *RW*. These steps are repeated. In section [10, 12), T_4 get one mandatory and T_5 get two. There still is one optional unit to be allocated. Since T_1 has the highest priority, it will be allocated in this section. Repeat till the last section. Finally, schedulable allocation is gotten as shown in Fig.2.

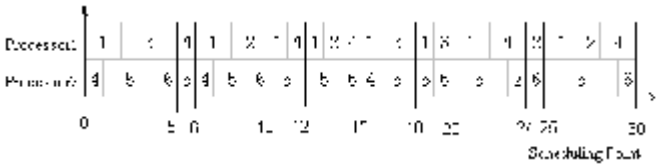

Fig.2. an allocation example of global scheduling algorithm

5. SIMULATION

In the simulations, task sets are generated automatically according to some initial parameters: the number of tasks, the minimum task period p_{min} and the maximum task period p_{max}.

Since the task set should meet the requirement $\sum u_i \leqslant U$ (U is the system utilization; u_i is the utilization of the i^{th} task in the set), so the more tasks a set includes, the less utilization each task can get. For each task, its c_i, p_i, r_i and d_i are generated randomly only if it meets the constrains below:

- $p_{min} \leqslant p_i \leqslant p_{max.}$
- $c_i \leqslant d_i - r_i$
- $u_i \leqslant 1$

The p_{min} and p_{max} is set because the relation of scheduling overhead and $\triangle p$ (= p_{max} - p_{min})will be investigated in experiments.

The simulation is executed using 4 clusters each equipped with 2 processors. The objective of experiments includes:

- Evaluating the scheduling overhead varying with the number of tasks.
- Evaluating the system utilization varying with the number of split tasks
- Evaluating the success ratio varying with the system utilization

5.1 Scheduling Overhead

This experiment investigates the relation between the numbers of scheduling points with the numbers of tasks. Three task sets are generated with the following setting:

- p_{min} =10, p_{max} =100, so $\triangle p$=90.
- p_{min} =50, p_{max} =100, so $\triangle p$=50.
- p_{min} =80, p_{max} =100, so $\triangle p$=20.

For each task set, number of tasks is added from 10 to 55.

Fig.3. demonstrates scheduling overhead varying with the increasing number of tasks. The x-axis records the number of tasks assigned to the cluster. The y-axis demonstrates the number of scheduling points. The result shows that the scheduling overhead goes up with an increasing rate. This is because the number of split tasks increases when cluster is almost full. And split tasks contribute to the increasing of *Lcm* (***T***) which will greatly impact the number of scheduling points.

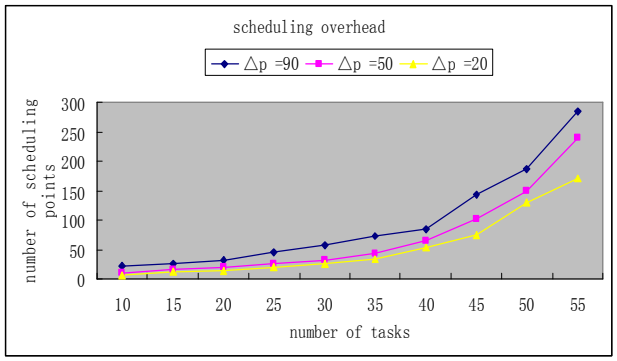

Fig.3. scheduling overhead varying with number of tasks

Furthermore, The figure shows that the number of scheduling points decreases with the dropping of $\triangle p$. This is because that more tasks are likely to have the same period when the $\triangle p$ decreasing, which makes the number of scheduling points less. Therefore, the case with $\triangle p$=20 has a lower scheduling overhead than that with $\triangle p$=90.

5.2 System Utilization

One of objectives of the paper is to make full use of system utilization. So this experiment is to investigate how much system utilization can be gotten by keeping adding tasks. Here, the three test sets that are the same as the experiment above are generated.

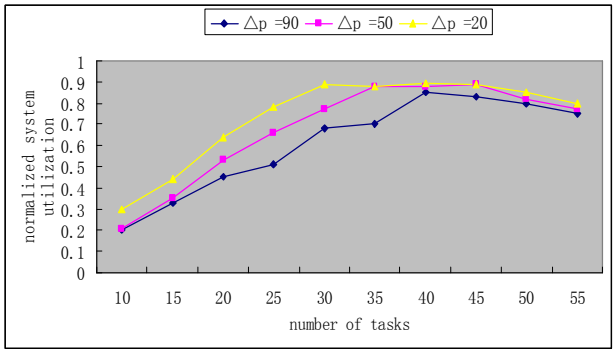

Fig.4. system utilization varying with number of tasks

Fig.4 demonstrates the result. The x-axis denotes the number of tasks assigned to the cluster. The y-axis records the normal system utilization. The normal system utilization is defined as $U/(m \cdot k)$, in which U means the system utilization and $m \cdot k$ is the number of processors.

The figure shows that the normal system utilization goes up and can be achieved more than 80% in the case of $\triangle p$ = 90. And better result can be achieved if $\triangle p$ =20. This is because the decreasing of $\triangle p$ results to the reducing of scheduling point, which contributes to the decreasing of context switch. This makes the system utilization is improved.

However, after that, for each case, the utilization is decreased slowly. This is because when the system is almost full, task has to be split if it still wants to be added into system. But the more split tasks the system has, the more contexts switch and task migration happens, which finally reduces the system utilization more or less. Therefore, the system cannot be fully used to executing tasks.

5.3 Success Ratio

Besides system utilization, the number of tasks can be successful scheduling is another point to be evaluated. The success ratio is defined as below:

Success Ratio = number of successful tasks / total number of tasks

The successful task means the task does not miss its deadline. So if the success ratio equals 1, we know that there are no tasks missing their deadline once they are successfully assigned to system.

The object of the experiment is to investigate how much the success ratio can be gotten if the system utilization are kept increasing.

Fig.5 demonstrates the result. The x-axis denotes the normalized system utilization. The y-axis records the success ratio. The result shows that the success ratio almost equals to 1 before the normalized system utilization goes to 75%. Then it goes down quickly with the increasing of system utilization. This is because the split tasks are increased greatly when the utilization surpasses 75%.

Furthermore, the less the $\triangle p$ is, the more smooth the success ratio drops after threshold. This is because the number of scheduling points of $\triangle p$ =20 is less than that of $\triangle p$ = 90, which counteracts part of negative effects brought by split tasks.

As a result, although the strategy of task splitting can increase the system utilization, it has an adverse impact on the success ratio. As a conclusion, if the system utilization over 75%, chasing for system utilization by splitting task is not a best choose. Besides, experiments shows that the small value of $\triangle p$ has a positive effects on the system utilization and success ratio.

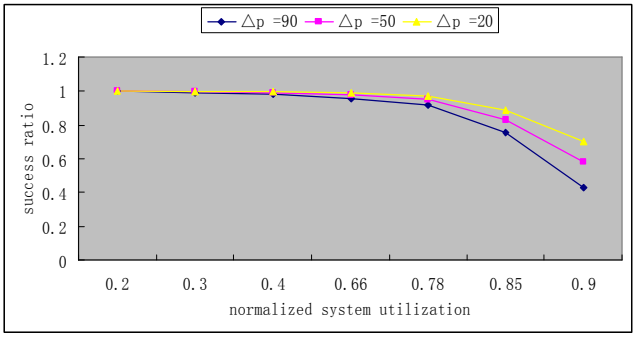

Fig.5. success ratio varying with system utilization

6. CONCLUSIONS

In this paper, real-time scheduling algorithms which tailor for Cyber-Physical Systems are proposed. The algorithms take real-time issue as well as system utilization, scheduling overhead and success ratio into account. Although the task assignment algorithm is not optimal, the time complex can be decreased to polynomial. Besides our task splitting algorithm takes the communication cost between clusters into account, which is never considered in prior works. At last, the global scheduling algorithm fits for the situation when splitting tasks exist in a task set.

The simulation results demonstrate that with the increasing number of tasks, the scheduling overhead is risen. Besides, the system utilization can be improved if we use the task splitting strategy. However, the success ratio decreases greatly when the system utilization is more than 75%.

As a conclusion, task splitting is a good way to improve the system utilization, but when the system utilization is more than 75%, the success ratio decreases sharply because too much split tasks are generated. So if the system utilization surpasses 75%, this solution of splitting task is a not good choice.

ACKNOWLEDGEMENTS

The work is supported by the 973 project (NO. 2011CB302802) and the Natural Science Foundation of China (NO. 61021004).

REFERENCES

Baruash, S.K., Cohen, N.K., Plaxton, C. G., Varvel, D.A. (1996). Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*. Volumn(15), pp. 600-625.

Benveniste, A. (2010). Loosely Time-Triggered Architectures for Cyber-Physical Systems. *EDDA 2010.* Campus de Beaulieu, 35042 Rennes cedex, France. 978-3-9810801-6-2/DATE10.

Gokhale, A., McDonale, M., Drager, S., and Mckeever, W. (2010). A Cyber Physical Systems Perspective on the Real-time and Reliable Dissemination of Information in Intelligent Transportation Systems. *Network Protocols and Algorithms*, Volume(2), ISSN 1943-3581.

Huang, H.M., Tidwell, T., Gill, C., and Lu, C. (2010). Cyber-Physical Systems for Real-Time Hybrid Structural Testing: A Case Study. Washington University, USA. *ICCPS'10, April 13-15, 2010, Stockholm, Sweden.*

Kang, K.D. and Son, S.H. Real-Time Data services for Cyber Physical Systems. *ICDCS Workshops,* pp.483-488. Department of Computer Science, State Univeristy of New York at Binghamton.

Koubaa, A. and Andersson, B. (2009). A vision of Cyber-Physical Internet. *CISTER Research Unit, Polytechnic Institute of Porto (ISEP/IPP).*

Lee, E.A. (2006). Cyber-Physical Systems-Are computing Foundations Adequate?. *In Proc. of NSF Workshop on Cyber-Physical System*.

Lee, E.A. (2008). Cyber Physical Systems: Design Challenges. Center for Hybrid and Embedded Software Systems, EECS. University of California, Berkeley. *In Proc. of ISORC,* May 2008. pp. 363-369.

Mollison, M.S., Erickson, J.P., Anderson, J.H., Baruah, S.K., and Scoredos, J.A. (2010). Mixed-Criticalit Real-Time Scheduling for Multicore Systems. *ICESS 2010.*

Mueller, F. (2006). Challenge for Cyber-Physical Systems: Security, Timing Analysis and Soft Error Protection. *HCSP-CPS, Nov 2006.* Department of Computer Science, North Carolina State University, Raleigh, NC.

Rajkumar, R. (2007). CPS briefing. *NSF,* May 10, 2007. Carnegie Mellon University.

Rajkumar, R., Lee, I., Sha, L., and Stankovic, J. (2010). Cyber-physical Systems- The next computing revolution. *In Proc. of Design Automation Conference,* 2010, Anaheim, California, USA.

Sha, L., Gopalakrishnan, S., Liu, X., Wang, Q. (2008). Cyber-physical systems: a new frontier. *IEEE*

International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing. DOI 10.1109/SUTC.2008.85.

Sha, L. and Meseguer, J. (2008). Design of Complex Cyber Physical System with Formalized Architectural Patterns. *Software-Intensive Systems and new computing Paradigms*. Springer, Nov, 2008. pp. 92-100. DOI: 10.1007/978-3-540-89437-7_5.

Talcott, C. (2008). Cyber-Physical Systems and Events. *Software-Intensive Systems, LNCS 5380*. SRI International, Menlo Park, CA 94025-3493, USA.

Tan, Y., Vuran, M.C., Goddard, S., Yu, Y., Song, M., and Ren, S. (2010). A concept Lattice-based Event Model for Cyber-Physical Systems. *ICCPS'10*, Stockholm, Sweden.

Tidwell, T., Glaubius, R., Gill, C.D., and Smart, W.D. (2010). Optimal Time Utility Based Scheduling Policy Design for Cyber-Physical Systems. Washington University in St. Louis.

Zhang, F., Shi, Z., and Wolf, W. (2009). A Dynamic Battery Model for Co-Design in Cyber-Physical Systems. *29th IEEE International Conference on Distributed Computing Systems Workshops*. ISBN: 978-0-7695-3660-6.

Zhang, F., Szwaykowska, K., Wolf, W., and Mooney V.J. (2008). Task Scheduling for Control Oriented Requirements for Cyber-Physical Systems. *RTSS, Real-Time Systems Symposium*.

Zhang, Y., Gill, C., and Lu, C. (2008). Reconfigurable Real-Time Middleware for Distributed Cyber-Physical Systems with Aperiodic Events. *ICDCS, 2008*. Dpeartment of Computer Science and Engineering, Washington University, St. Louis, MO, USA.

Zhu, D.K., Mosse, M., and Melhem, R. (2003). Multiple-Resource Periodc Scheuling Problem: how muchfairness is necessary? *In Proc. Of the 24th IEEE Real-Time System Symposium*.

Zimmer, C., Bhat, B., Mueller, F., and Mohan, S. (2010). Time-Based Intrusion Detection in Cyber-Physical Systems. *ICCPS'10*, Stockholm, Sweden.

Appendix A. EXAMPLE OF ALGORITHM 3

Table 1. The execution example of algorithm 3

time	0	5	6	10	12	15	18	20	24	25	30
b_k	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}
RW_1^k	*	0	2/5	0	-1/5	0	1/5	0	-2/5	0	0
RW_2^k	*	1	6/5	0	2/5	0	3/5	1	4/5	0	0
RW_3^k	*	0	*	*	*	*	-1/5	0	*	*	0
RW_4^k	*	2/3	0	1/3	0	0	0	2/3	0	1/3	0
RW_5^k	*	1/3	0	2/3	0	0	0	1/3	0	2/3	0
RW_6^k	*	0	1/5	0	2/5	0	3/5	0	4/5	0	0
m_1^k	*	2	0	2	0	1	1	1	1	0	2
m_2^k	*	0	0	2	0	1	0	0	1	1	1
m_3^k	*	3	*	*	*	*	1	1	*	*	*
m_4^k	*	1	1	1	1	1	1	0	2	0	2
m_5^k	*	3	1	2	2	2	2	1	3	0	4
m_6^k	*	1	0	1	0	1	0	1	0	1	1
PW_1^k	*	0	2/5	0	4/5	0	1/5	0	3/5	0	0
PW_2^k	*	1	6/5	4/5	2/5	0	3/5	1	4/5	0	0
PW_3^k	*	0	*	*	*	*	4/5	0	*	*	0
PW_4^k	*	2/3	0	1/3	0	0	0	2/3	0	1/3	0
PW_5^k	*	1/3	0	2/3	0	0	0	1/3	0	2/3	0
PW_6^k	*	0	1/5	0	2/5	0	3/5	0	4/5	0	0
$\alpha_k(T_1)$	*	-	-	-	-	-	-	-	0	*	*
$\alpha_k(T_2)$	*	-	-	-	-	-	-	-	0	*	*
$\alpha_k(T_3)$	*	-	*	*	*	*	0	-	*	*	*
$\alpha_k(T_4)$	*	0	-	-	-	-	-	-	-	*	*

$\alpha_k(T_5)$	*	0	-	0	-	-	-	-	-	*	*
$\alpha_k(T_6)$	*	-	-	-	-	-	-	-	0	*	*
UF_1^k	*	5/2	3/2	*	1/2	5/2	*	5/2	*	*	*
UF_2^k	*	5	4	*	3	5	*	5	*	*	*
UF_3^k	*	5/3	*	*	*	*	*	*	*	*	*
UF_4^k	*	0	3	*	*	3	*	3	*	*	*
UF_5^k	*	0	3/2	*	*	3/2	*	3/2	*	*	*
UF_6^k	*	5	4	*	3	5	*	5	*	*	*
o_1^k	*	*	*	0	1	*	0	*	*	*	*
o_2^k	*	*	*	0	0	*	0	*	*	*	*
o_3^k	*	*	*	0	0	*	1	*	*	*	*
o_4^k	*	*	*	0	0	*	0	*	*	*	*
o_5^k	*	*	*	0	0	*	0	*	*	*	*
o_6^k	*	*	*	0	0	*	0	*	*	*	*