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
Department of Computer Science, University of Warwick
Coventry, United Kingdom, CV4 7AL
{liganghe, saj, dps, grn}@dcs.warwick.ac.uk

Abstract

In this research, a scenario is assumed where periodic real-time jobs are being run on a heterogeneous cluster of computers, and new aperiodic parallel real-time jobs, modelled by Directed Acyclic Graphs (DAG), arrive at the system dynamically. In the scheduling scheme presented in this paper, a global scheduler situated within the cluster schedules new jobs onto the computers by modelling their spare capabilities left by existing periodic jobs. Admission control is introduced so that new jobs are rejected if their deadlines cannot be met under the precondition of still guaranteeing the real-time requirements of existing jobs. Each computer within the cluster houses a local scheduler, which uniformly schedules both periodic job instances and the subtasks in the parallel real-time jobs using an Early Deadline First policy. The modelling of the spare capabilities is optimal in the sense that once a new task starts running on a computer, it will utilize all the spare capability left by the periodic real-time jobs and its finish time is the earliest possible. The performance of the proposed modelling approach and scheduling scheme is evaluated by extensive simulation; results show that the system utilization is significantly enhanced, while the real-time requirements of the existing jobs remain guaranteed[*].

1. Introduction

Cluster systems are gaining in popularity for processing scientific and commercial applications [6]. They are also increasingly used for processing applications with time constraints [1] as the work has been done to extend conventional operating systems, such as Linux, to support real-time scheduling (for example, the preemptive scheduling based on the Earliest-Deadline-First policy) [5][16]. Real-time processing can often be represented abstractly as the hybrid execution of existing periodic jobs together with newly arriving aperiodic jobs. An example of this is in the reservation-based scheduling of multimedia applications, where the reservation of processor times can be expressed per period, so as to ensure that the processor utilization for an application is maintained above a desired level [9]. These reserved executions can be viewed as periodic jobs and besides the reserved executions, the processors have also to deal with other newly arriving jobs. This scenario presents the challenge of devising scheduling schemes which judicially deal with the hybrid execution of existing jobs (or reserved executions) together with newly arriving jobs. This task is complicated when the objective is to reduce the response times of newly arriving jobs while maintaining the time constraints of existing periodic jobs.

The dynamic scheduling technique presented in this paper addresses this issue, aiming to allocate newly arriving *Aperiodic Real-time Jobs* (ARJ) to a heterogeneous cluster of computers on which *Periodic Real-time Jobs* (PRJ) are running. In this paper, an ARJ is assumed to be a parallel job with time constraints, which is modeled as a real-time *Directed Acyclic Graph* (DAG) [11]. In this scheduling framework, a global scheduler located on one of the computers in the heterogeneous cluster analyzes the execution of PRJs in the remaining computers and models the initial distribution of their spare capabilities off-line. When a new ARJ arrives at the system dynamically, the global scheduler releases the precedence constraints among the tasks in the ARJ, adjusts the initial distribution of spare capabilities on-line (which may be altered due to the dynamic arrivals of the preceding ARJs) and tries to fit the execution of tasks in the new ARJ into spare time slots in the computers. The global scheduling for ARJs takes both task and message scheduling into account. In the local scheduling (at each computer), a uniform Early Deadline First (EDF) policy is used for both the PRJ in-

[*] This work is sponsored in part by grants from the NASA AMES Research Center (administrated by USARDSG, contract no. N68171-01-C-9012), the EPSRC (contract no. GR/R47424/01) and the EPSRC e-Science Core Programme (contract no. GR/S03058/01).

stances and the tasks in the ARJs so as to reduce the local scheduling complexity.

The approach for modelling spare capabilities proposed in this paper does not invoke any communication overhead between the global scheduler and the remaining computers in the cluster. The approach is optimal in the sense that once a new task starts running, it will utilize all the spare capability left by the PRJs, and its finish time is the earliest possible.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 describes the workload and system model. In section 4, a novel approach is presented to enable the global scheduler to model spare capabilities of computers in a cluster. Section 5 proposes a global dynamic scheduling algorithm for parallel real-time jobs. The performance evaluation is presented in Section 6 and Section 7 concludes the paper.

2. Related work

Studies on heterogeneous clusters or networks of workstations have received a good deal attention [7][17]. The scheduling of tasks with precedence constraints, which are usually represented by *Directed Acyclic Graphs* (DAG), has also been documented in a number of papers [2][11][19][20]. An off-line algorithm is presented in [2] to schedule communicating tasks with precedence constraints in distributed systems. However, the algorithm belongs to the static category. Paper [20] describes a run-time incremental DAG scheduling approach on parallel machines. The approach is however limited to homogenous systems. Two low-complexity heuristics, the Heterogeneous Earliest-Finish-Time Algorithm and the Critical-Path-on-a-Processor Algorithm are proposed in [19] for scheduling DAGs on heterogeneous processors. However, the heuristics are designed for non-real-time task allocation. In [11] non-real-time DAGs are extended to include real-time information, and the scheduling of parallel tasks with real-time DAG topologies onto heterogeneous systems is proposed. This technique however is not aimed at using spare system capabilities. The scheduling scheme presented in this paper dynamically schedules *parallel real-time jobs* with real-time DAGs on heterogeneous clusters. Both task scheduling and message scheduling are taken in account in the algorithm design.

A number of scheduling algorithms for periodic real-time jobs on multi-computer or multiprocessor systems have also been presented [3][12]. A task duplication technique combined with pipelined execution is presented in [12], allowing the scheduling of time critical periodic applications on heterogeneous systems. [3] addresses the reward-based scheduling problem for periodic tasks, which assumes that a periodic task comprises a mandatory and an optional part. While these techniques are effective, they are unable to deal with the hybrid execution of periodic and aperiodic tasks.

Scheduling systems for processing both periodic and aperiodic real-time tasks can be classified into fixed or dynamic priority systems. Dynamic priority systems typically attain higher processor utilization than fixed ones. Slack Stealing policies have been designed for fixed priority systems [8] while the Background (BG), Deadline Deferrable Server (DDS), Total Bandwidth Server (TBS) and Improved Priority Exchange (IPE) algorithms have been designed for dynamic priority systems [4][13][15]. These techniques are widely used in embedded real-time systems, such as robot control systems. All these algorithms have been developed for uniprocessor architectures and aperiodic tasks are assumed independent without precedence constraints. The techniques presented in [8] and [18] run aperiodic tasks by using the spare capability left by periodic tasks. Unfortunately, such schemes are limited to uniprocessor scenarios.

It is a non-trivial task to extend the modelling of spare capability from uniprocessor architectures to cluster environments. A uniprocessor system only models the spare capability in itself. In a cluster scenario however, a central node models the spare capabilities of other nodes, so the information needed for calculating spare capabilities is far more difficult to attain. An efficient modelling approach for clusters that avoids significant communication overheads among nodes is therefore needed. The modelling approach in this paper is able to model spare capability of computers in a heterogeneous cluster and is free of additional communication overheads. The scheme in this paper is also designed for dynamic priority systems and an EDF policy is used in the local scheduling.

3. Workload and system model

A heterogeneous cluster of computers is modeled as $P=\{p_1, p_2,..., p_m\}$, where p_i is an autonomous computer. Each computer p_i is weighted pw_i, which represents the time it takes to perform one unit of computation. The computers in the heterogeneous cluster are connected by a multi-bandwidth local network. Each communication link between computer p_i and p_j, denoted by l_{ij}, is weighted lw_{ij}, which models the time it takes to transfer one unit of message between p_i and p_j.

Each computer runs a set of PRJs, all of which are independent of one another. On a computer with n PRJs, the i-th periodic real-time job PRJ_i ($1 \le i \le n$) is defined as (S_i, C_i, T_i), where S_i is PRJ_i's start time, C_i is PRJ_i's execution time on the computer, and T_i is PRJ_i's period. An execution of PRJ_i is called a *Periodic Job Instance* (PJI) and the j-th execution is denoted by PJI_{ij}. PJI_{ij} is ready at time $(j-1)*T_i$, termed the *ready time* (R_{ij}, $R_{i1}=S_i$), and must be complete before $j*T_i$, termed the *deadline* (D_{ij}). All PJIs

must meet their deadlines and are scheduled using an EDF policy. Fig.1 shows two PRJs and their execution on a single computer; all the illustrations in this paper use these two PRJs as a working example.

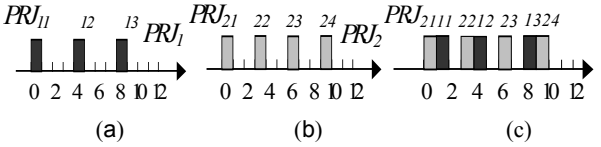


Figure 1. **A case study of PRJs (a) PRJ_1 with a period of 4 and an execution time of 1, (b) PRJ_2 with a period of 3 and an execution time of 1, (c) execution of PRJ_1 and PRJ_2 under EDF**

ARJs arrive at the heterogeneous cluster dynamically. If accepted, an ARJ is run once. An ARJ is modeled as (*avt*, *J*), where *avt* is the ARJ's arrival time and *J* defines the tasks and their topology in the ARJ. $J=\{V, E\}$, where $V=\{v_1, v_2, \ldots, v_r\}$, which defines *r* real-time tasks that constitute the ARJ. $dt(v_i)$ and cv_i are denoted as v_i's deadline and computational volume; *E* represents the communication relationship and the precedence constraints among tasks; $e_{ij}=(v_i, v_j)\in E$ represents a message sent from task v_i to v_j and it also suggests v_j can start running only after v_i is complete and v_j receives message e_{ij}; v_i is called v_j's predecessor and mv_{ij} is denoted as message e_{ij}'s volume.

Fig.2 depicts the components of the scheduler model in the heterogeneous cluster environment. It is assumed that PRJs are active across the constituent computers, and a central computer in the cluster, the *global scheduler*, records S_i, C_i and T_i of all PRJs. The global scheduler models the cluster spare capacities left by the PRJs.

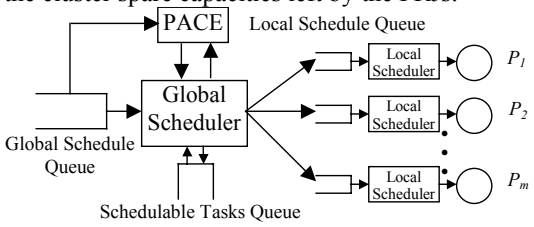


Figure 2. **The scheduler model in the heterogeneous cluster environment**

When the global scheduler fetches a newly arriving ARJ from the head of the global schedule queue, it inserts the schedulable tasks in the ARJ into the schedulable task queue. The global scheduler then picks a task from the head of the schedulable task queue and schedules it globally. Once an ARJ is accepted, tasks in the ARJ are sent to the local schedulers of the designated computers. At each computer, the local scheduler uniformly schedules both the ARJs' tasks and the PRJs' PJIs using EDF. The local schedule is preemptive.

PACE is a performance prediction toolkit [10]. In this scheduler model, PACE accepts ARJs, predicts the execution time of each task in the ARJs on each computer and returns the predicted time to the global scheduler. After the global scheduler decides to schedule task v_i on computer p_s, assuming message $e_{ij}=(v_i, v_j)\in E$, PACE is called to predict e_{ij}'s communication time on each link between p_s and any other computer. The efficiency of the PACE evaluation engine enables the real-time production of performance data [14].

4. Spare capability modelling

In this section, the initial distribution of idle time left by the PRJs running on the computers is modelled. The idle time distribution will be altered by the dynamic arrivals of ARJs. Hence, an on-line mechanism is presented for adjusting the initial idle time distribution when the global scheduler schedules a newly arriving ARJ.

4.1. Off-line modelling of the initial distribution of spare capabilities

As an example, consider the two PRJs found in Fig.1 that are mapped to a single computer. Consider the case for PRJ_1 where there are 4 time units before PJI_{11}'s deadline and there are two tasks, PJI_{11} and PJI_{21}, which must be completed before that time. There are therefore 2 idle time units before PJI_{11}'s deadline. In the case of PRJ_2 there are 6 time units before PRJ_{22}'s deadline and 3 tasks, PJI_{11}, PJI_{21} and PJI_{22}, which must be completed before that time. In this case, 3 idle time units are available before the deadline.

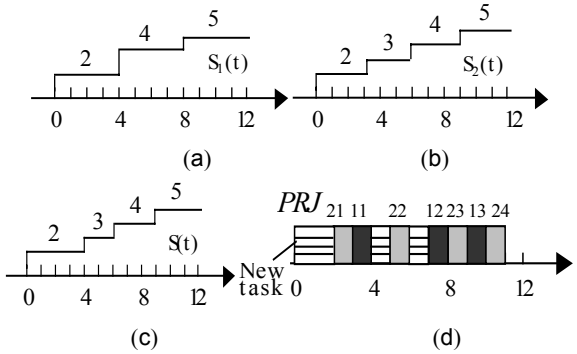


Figure 3. **A case study of the function of idle time units (a) Function of idle time units for PRJ_1 (b) Function of idle time units for PRJ_2 (c) Function of idle time units for both PRJ_1 and PRJ_2 (d) The joint execution of PRJ_1, PRJ_2 and a new task with an execution time of 4 starting at 0**

The above calculation can be performed for all *PJI*s of any PRJ_i running on the same computer, and a function constructed of idle time units corresponding to PRJ_i, denoted as $S_i(t)$, is defined in Eq.1, where D_{ij} is PJI_{ij}'s deadline (let D_{i0} be 0), P_{ij} is the sum of execution time of *PJI*s that must be completed before D_{ij}.

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

P_{ij} can be computed as Eq.2 (S_k is PRJ_k's start time).

$$P_{ij} = \sum_{k=1}^{n} \lfloor \alpha / T_k \rfloor * C_k \text{, where, } \alpha = \begin{cases} D_{ij} - S_k & D_{ij} > S_k \\ 0 & D_{ij} \le S_k \end{cases} \quad (2)$$

Fig.3.a and Fig.3.b show the functions of idle time units within a certain time period, $S_1(t)$ and $S_2(t)$, corresponding to PRJ_1 and PRJ_2 in Fig.1 respectively. In the figures, the time points, except zero, at which the function value increases, are called *Jumping Time Points* (*JTP*). A *JTP* is a *PJI*'s deadline. In Fig.3.a, the *JTP*s are 4 and 8. If the number of time units that are used to run new tasks between time 0 and any *JTP* is less than $S_i(JTP)$, the deadlines of all PJIs of PRJ_i can be guaranteed.

Suppose n PRJs (PRJ_1,..., PRJ_i,..., PRJ_n) are running on a single computer, then the distribution function of idle time left by the PRJs, denoted as $S(t)$, can be derived from the individual $S_i(t)$ ($1 \le i \le n$). For any time t, $S(t)$ obtains its value from the minimum of all $S_i(t)$, shown in Eq.3.

$$S(t) = min\{S_i(t) | 1 \le i \le n\} \quad (3)$$

*JTP*s are also defined in $S(t)$, as with $S_i(t)$. $S(t)$ suggests that idle time units of $S(JTP)$ are available in [0, *JTP*]. Thus, in order to satisfy the real-time requirements of all PRJs, for any *JTP*, at most $S(JTP)$ time units can be used to run new tasks in [0, *JTP*]. The initial distribution of spare capabilities in each computer is constructed off-line.

$S(t)$ corresponding to the *PRJ* set consisting of PRJ_1 and PRJ_2, is plotted in Fig. 3.c. Fig.3.d illustrates the execution of a new task in which the real-time requirements of PRJ_1 and PRJ_2 is still guaranteed. The execution coincides with function $S(t)$ in Fig.3.c, that is, between time 0 and any *JTP* there are exactly S(*JTP*) time units used to run the new task.

4.2. On-line modelling of the spare capability distribution

If a new task starts running at any time t_0, the number of idle time units in [t_0, *JTP*] ($t_0 < JTP$), denoted by $S(t_0, JTP)$, needs to be calculated on-line. In order to do this, it is necessary to calculate the proportion of workload that all PJIs which are to complete in [0, *JTP*] have finished before t_0, and also how much finishes in [t_0, *JTP*]. The remaining time in [t_0, *JTP*] will then be spare. This calculation involves identifying what PJIs must be complete before t_0, and what PJIs can start before t_0 but must be complete before the *JTP*. Some notation is introduced below to classify the PJIs.

$PJ(t_0)$ is a set of PJIs whose deadlines are no more than time t_0. Hence, all PJIs in $PJ(t_0)$ must be complete before t_0. $PJ(t_0)$ is defined in Eq.4.

$$PJ(t_0) = \{PJI_{ij} | D_{ij} \le t_0\} \quad (4)$$

$P(t_0)$ are denoted as the number of time units in [0, t_0] that are used for running the PJIs in $PJ(t_0)$. $P(t_0)$ can be calculated by Eq.5.

$$P(t_0) = \sum_{k=1}^{n} \lfloor \alpha / T_k \rfloor * C_k \text{, where, } \alpha = \begin{cases} t_0 - S_k & t_0 > S_k \\ 0 & t_0 \le S_k \end{cases} \quad (5)$$

Let JTP_1, JTP_2,..., JTP_k be a sequence of *JTP*s after t_0 in the spare capability distribution function $S(t)$ and JTP_1 the nearest to t_0. $LJ_k(t_0)$ is a set of *PJI*s, whose ready times are less than t_0, and whose deadlines are more than t_0 but no more than JTP_k. $LJ_k(t_0)$ is defined in Eq.6. All *PJI*s in $LJ_k(t_0)$ can start running before t_0 but must be complete before JTP_k. $L_k(t_0)$ is denoted as the number of time units in [0, t_0] that are used to run the *PJI*s in $LJ_k(t_0)$.

$$LJ_k(t_0) = \{ PJI_{ij} | R_{ij} < t_0 < D_{ij} \text{ and } D_{ij} \le JTP_k\} \quad (6)$$

In Theorem 1, $S(t_0, JTP_k)$ is related to $S(JTP_k)$. $S(JTP_k)$ is obtained directly from the initial spare capability distribution function established off-line in subsection 4.1.

Theorem 1. Suppose t_0 is any time point in [0, JTP_k], then $S(JTP_k)$ and $S(t_0, JTP_k)$ satisfy the following equation:

$$S(t_0, JTP_k) = S(JTP_k) - t_0 + P(t_0) + L_k(t_0) \quad (7)$$

Proof: *PJI*s whose deadlines are less than JTP_k must be completed in [0, JTP_k]. Their total workload is $P(JTP_k)$ (see Eq.5). The workload of $P(t_0)$ and $L_k(t_0)$ has to been finished before t_0, so the workload of

$$P(JTP_k) - P(t_0) - L_k(t_0)$$

must be done in [t_0, JTP_k]. Hence, the maximal number of time units that can be spared to run new tasks in [t_0, JTP_k], i.e. $S(t_0, JTP_k)$, is $(JTP_k - t_0) - (P(JTP_k) - P(t_0) - L_k(t_0))$. Thus, the following equation exists:

$$S(t_0, JTP_k) = JTP_k - P(JTP_k) - t_0 + P(t_0) + L_k(t_0)$$

In addition, $JTP_k - P(JTP_k) = S(JTP_k)$. Hence Eq.7 holds. □

$L_k(t_0)$ in Eq.7 still remains unknown. The rest of the subsection documents the calculation of $L_k(t_0)$.

If there are new tasks running before t_0, the execution process of PJIs in $PJ(t_0)$ may change so that they may not retain the same execution pattern as the case when PRJs alone are executing. Theorem 2 is introduced to reveal the distribution property of the remaining time units before t_0 after running PJIs in $PJ(t_0)$ as well as the new tasks.

Theorem 2. Suppose the last executed new task is completed at time f, then there exists such a time point t_s in [f, t_0] ($t_0 > f$), where

1. either PJIs in $PJ(t_0)$ retain the same execution pattern in [t_s, t_0] as the case when no new tasks are run before t_0, or all PJIs in $PJ(t_0)$ are completed before t_s,
2. there are no idle time slots in [f, t_s],
3. t_s can be determined by Eq.8, where $I_p^{t_0}(t_s, t_0)$ represents the number of time units left in [t_s, t_0] after executing PJIs in $PJ(t_0)$; $I_{P,A}^{t_0}(f, t_0)$ represents the number of time units left in [f, t_0] after executing both PJIs in $PJ(t_0)$ and also the new tasks.

$$I_p^{t_0}(t_s, t_0) = I_{P,A}^{t_0}(f, t_0) \quad (8)$$

Proof: The execution of the last new tasks may delay the execution of PJIs in $PJ(t_0)$. The delayed PJIs may also delay other PJIs in $PJ(t_0)$ further. The delay chain will however cease when the delayed PJIs no longer delay other PJIs, or all the PJIs in $PJ(t_0)$ are complete. Since all PJIs

$PJ(t_0)$ must be complete before t_0, such a time point, t_s, must exist that satisfies Theorem 2.1. Since there are unfinished workloads before t_s, Theorem 2.2 also exists. Eq.8 is a direct derivation from Theorem 2.1 and 2.2. □

Theorem 2 is illustrated by comparing the PJIs' execution in Fig.1.c and 3.d. In Fig.1.c, PJI_{12} and PJI_{23} finish at time 5 and 7 respectively. Due to the execution of the new task, PJI_{12} and PJI_{23} are delayed to finish at times 8 and 9, respectively, shown in Fig.3.d. PJI_{23}'s delay, further delays PJI_{13}, and PJI_{24} is then delayed by PJI_{13}. In Fig.3.d however, PJIs ready after time 11 can be run without further disruption. Time 11 can be set as t_s in the example.

As shown in Theorem 2, PJIs in $PJ(t_0)$ running in $[t_s, t_0]$ retain the original execution pattern (as though there were no preceding new tasks). Hence the remaining time units in $[t_s, t_0]$ after running these PJIs can be calculated; these time units can only be occupied by PJIs in $LJ_k(t_0)$. Consequently, $L_k(t_0)$ in Eq.7 can be calculated. The algorithm for computing $L_k(t_0)$ is omitted in this paper.

5. Scheduling algorithm

Let v_i be a task in an ARJ. Denote $st^k(v_i)$ and $ft^k(v_i)$ as task v_i's earliest possible start time and its finish time on computer p_k. It is assumed that tasks $v_{i1}, v_{i2},\ldots, v_{iq}$ (v_{iq} is the last task) have been scheduled on p_k. $st^k(v_i)$ can be calculated using Eq.9, where, $mlt^k(v_i)$ is the latest time when all messages from v_i's predecessors arrive at p_k.

$$st^k(v_i)=\begin{cases}\max(mlt^k(v_i), ft^k(v_{iq})) & v_i \text{ has predecessors}\\ \max(avt, ft^k(v_{iq})) & otherwise\end{cases} \quad (9)$$

Suppose v_i is scheduled on computer p_k. The arrival time of the message from v_i's predecessor v_j to v_i (i.e. message e_{ji}) is denoted by $mat^k(v_j, v_i)$. If v_j is also scheduled on p_k, then $mat^k(v_j, v_i)$ equals $ft^k(v_j)$. Suppose v_j is scheduled on p_s ($s\neq k$) and there exists a message schedule sequence, (mst_1^{sk}, mft_1^{sk}), (mst_2^{sk}, mft_2^{sk}),…, (mst_a^{sk}, mft_a^{sk}), in the communication link between p_s and p_k, where mst_i^{sk} and mft_i^{sk} are the starting time and finish time of a message transferring in the communication link; then the first idle time slot in the communication link satisfying Eq.10 is used to send e_{ji}, where $com^{sk}(e_{ji})$ is the communication time of e_{ji} in the communication link between p_s and p_k; this idle slot is supposed to be $(mft_b^{sk}, mst_{b+1}^{sk})$.

$$mst_q^{sk}-\max(mft_{q-1}^{sk}, ft^s(v_j))\geq com^{sk}(e_{ji}) \quad (1\leq q\leq a+1, \text{ let } mft_0^{sk}=0,\ mst_{a+1}^{sk}=\infty) \quad (10)$$

Thus, $mat^k(v_j, v_i)$ can be calculated by Eq.11.

$$mat^k(v_j,v_i)=\begin{cases}\max(mft_b, ft^s(v_j))+com^{sk}(e_{ji}) & s\neq k\\ ft^k(v_j) & s=k\end{cases} \quad (11)$$

Then, $mlt^k(v_i)$ in Eq.9 can be calculated by Eq.12.

$$mlt^k(v_i)=max\{mat^k(v_j, v_i)|\ v_j \text{ is } v_i\text{'s predecessor}\} \quad (12)$$

The complete scheduling procedure for ARJs is as follows. The global scheduler fetches an ARJ from the head of the global schedule queue, and inserts the schedulable tasks in the ARJ (a task is schedulable if it either has no predecessors or all of its predecessors have been scheduled) into the schedulable task queue so that the deadlines of the tasks are in increasing order. The global scheduler then picks a task from the head of the queue and schedules it globally. The starting time of a task v_i is calculated as Eq.9. Suppose that v_i starts at t_0 on computer p_j, using Eq.7, the global scheduler can calculate in p_j how many idle time units there are between t_0 and any *JTP* following t_0, which can be used to run v_i. Therefore, it can be determined before which *JTP* v_i can be completed. Consequently, v_i's finish time at any computer can be determined. This is shown in Algorithm 1.

Algorithm 1. Calculating the finish time of task v_i starting at t_0 in computer p_j

1. $c^j(cv_i)\leftarrow v_i$'s execution time on p_j (predicted by PACE);
2. Calculate $P(t_0)$ using Eq.5; Get t_s using Eq.8;
3. Get the first *JTP* after t_0;
4. Call Algorithm 1 to calculate the corresponding $L_k(t_0)$;
5. Calculate $S(t_0, JTP)$ using Eq.7;
6. **while** ($S(t_0, JTP)<c^j(cv_i)$)
7. $OJTP\leftarrow JTP$; Get the next *JTP*;
8. Calculate $S(t_0, JTP)$ by Eq.7;
9. **end while**
10. $ft^j(v_i)\leftarrow OJTP+c^j(cv_i)-S(t_0, OJTP)$;

If v_i's finish time on any computer in the cluster is greater than its deadline, the ARJ that v_i belongs to is rejected. The admission control is shown in Algorithm 2.

Algorithm 2. Admission Control

1. PC$\leftarrow\Phi$;
2. **for** each computer p_j in the cluster **do**
3. Calculate v_i's starting time on p_j, $st^j(v_i)$, using Eq.9;
4. Call Algorithm 1 to calculate v_i's finish time on p_j, $ft^j(v_i)$;
5. **if** ($ft^j(v_i)\leq dt(v_i)$) **then**
6. PC=PC$\cup\{p_j\}$;
7. **end for**
8. **if** PC=Φ **then** reject v_i and the ARJ that v_i belongs to;
9. **else** accept v_i;

When v_i's deadline can be met in more than one computer, two possible *Second-level Selection Policies* are offered to choose a final computer. One is a *Response First* (RF) policy, which selects the computer on which v_i has the earliest finish time. The other is *Utilization First* (UF) policy, which selects the computer on which v_i has the longest execution time. The two policies have different selection inclinations. In section 6 the performance of these two policies is evaluated.

After deciding which computer the task v_i should be scheduled to, the global scheduler resets v_i's deadline to its finish time on that computer. If all tasks in an ARJ are accepted, these tasks are sent to the designated computers.

When the local scheduler in any computer receives the allocated ARJs' tasks or the PJIs of the PRJs are ready, it inserts them into the *Local Schedule Queue* ordered in increasing deadlines. A local scheduler fetches a task (ARJ's task or PJI) from the head of the queue and the task is then executed. Once the task with the lower deadline is ready, the current execution is preempted.

Assume that the initial distribution of spare capabilities on each computer in the heterogeneous cluster is constructed off-line. The on-line *global dynamic scheduling algorithm* (GDS) is shown in Algorithm 3.

Algorithm 3. Global dynamic scheduling for parallel real-time jobs

1. **if** global scheduler queue=Φ **then** wait until a new ARJ arrives, then go to step 3;
2. **else**
3. Get a job from the head of global scheduler queue and insert its schedulable tasks into the schedulable task queue;
4. **for** each task v_i in the schedulable task queue **do**
5. Call Algorithm 2 to exert admission control;
6. **if** accept v_i **then**
7. Call a second-level selection policy to choose a computer p_j;
8. Reset v_i's deadline to be its computed finish time $ft^j(v_i)$;
9. Search for new schedulable tasks in the ARJ and insert them into the schedulable task queue;
10. **else** go to step 1;
11. **end if**
12. **end for**
13. Dispatch the tasks in the ARJ to designated computers; go to step 1;
14. **end if**

Since v_i's deadline is reset to its finish time, v_i will be forced to run between its starting time and the deadline. As the modelling analysis suggests in Section 4, v_i cannot be finished earlier on the computer on which it is scheduled. Otherwise, the deadlines of some PJIs on that computer must be missed. In this sense, the modelling approach is optimal. When the global scheduler models spare capacities in other computers, no information has to be transferred among them in order to make scheduling decisions. Hence no communication overhead is incurred by the modelling approach.

6. Performance evaluation

The experimental parameters in our simulation studies are chosen either based on those used in the literature [8][11] or to represent a realistic workload.

Sets of 40 PRJs are randomly generated with periods ranging from 42 to 15015. The level of PRJ workloads (*PLOAD*) is set by varying PRJs' execution times. Three levels of *PLOAD*, light, medium and heavy, are generated for each computer, which provides 10%, 40% and 70% system utilization, respectively.

In the simulation, task v_i's execution time on computer p_j is calculated as $\lfloor cv_i * pw_j \rfloor$; similarly, message e_{ij}'s communication time on link l_{st} is $\lfloor mv_{ij} * lw_{st} \rfloor$. In a heterogeneous cluster, computer p_i's weight pw_i is uniformly chosen between *MIN_PW* and *MAX_PW*. This range reflects the level of computational heterogeneity. The weight of a communication link is uniformly chosen between *MIN_LW* and *MAX_LW*. This range reflects the level of communication heterogeneity.

Each point in the performance curve is plotted as the average value of the corresponding performance measure of 10,000 independent ARJs. ARJs are assumed to arrive following a Poisson process with an arrival rate λ. Each ARJ has a randomly generated DAG topology with a given number of tasks (*TASKNUM*); task v_i's computational volume cv_i is uniformly chosen between *MIN_CV* and *MAX_CV* and the volume of a message among tasks is uniformly chosen between *MIN_ MV* and *MAX_MV*. v_i's deadline is defined as follows: if v_i has no predecessors in the DAG, $dt(v_i)=avt+cv_i * \overline{nw} * (dr+1)$, where the parameter dr is uniformly chosen between *MIN_DR* and *MAX_DR*, and $\overline{nw}$ is the geometric mean of the weight of all computers; otherwise,

$$dt(v_i)=\max\{dt(v_j)\}+cv_i * \overline{nw} * (dr+1)$$

where v_j is v_i's predecessor.

Table 1 **Parameters for simulation studies**

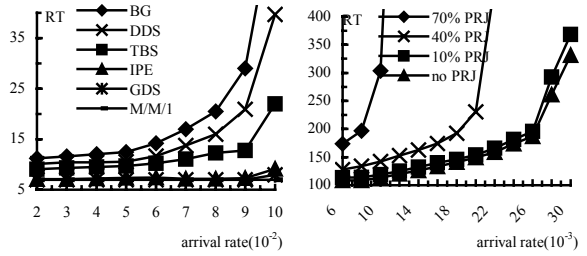
Parameter	Explanation	Value
MAX_PW/MIN_PW	Maximum/minimum computer weight	4.0/1.0
MAX_LW/MIN_LW	Maximum/minimum link weight	4.0/1.0
MAX_CV/MIN_CV	Maximum/minimum computation volume	25/5
MAX_MV/MIN_MV	Maximum/minimum message volume	5/1
MAX_DR/MIN_DR	Maximum/minimum deadline ratio	2.0/0
PLOAD	System utilization provided by PRJs	10%, 40%, 70%
PNUM	Computer number in a cluster	8
TASKNUM	Task number in an ARJ	16

The values of the simulation parameters are given in Table 1 unless otherwise stated. Three metrics are measured in the simulation experiments: *Guarantee Ratio* (GR), *System Utilization* (SU) and *average Response Time* (RT). The GR is defined as the percentage of jobs guaranteed to meet their deadlines. The SU of a cluster is

defined as the fraction of busy time for running tasks to the total time available in the cluster. An ARJ's response time is defined as the difference between its arrival time and the finish time of the last task to be run. RT is the average response time for all ARJs.

6.1. Job workloads and second-level selection policies

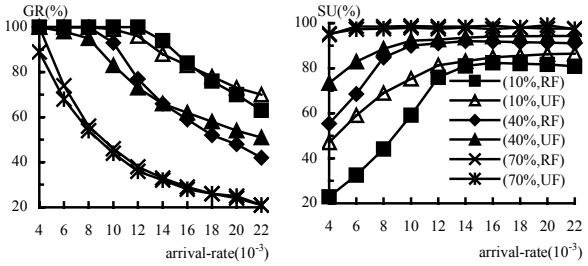
RT can be viewed as a measure of the capability of the scheme in utilizing the spare capability in the computers. Fig.4.a compares the *global dynamic scheduling algorithm* (GDS) presented in this paper in terms of RT with four other algorithms for dynamic priority systems in the literature [4][13][15], i.e. Background (BG), Deadline Deferrable Server (DDS), Total Bandwidth Server (TBS) and Improved Priority Exchange (IPE). It is noted that our GDS algorithm is devised for scheduling parallel real-time jobs in a cluster scenario, but all other algorithms are designed for scheduling periodic and independent aperiodic tasks (non-parallel tasks) in uniprocessor architectures. In order to make a fair comparison, in this experiment the GDS is downgraded to schedule independent real-time tasks in a cluster of two computers, one acting as the global scheduler, the other housing a local scheduler and jointly running tasks; the computational volumes of tasks follow the exponential distribution. To stress the response performance, the GR of non-periodic real-time tasks is fixed to be 1.0 by assigning extremely loose deadlines. An M/M/1 queuing model is used to compute the ideal bound for RT of the same workload but in the absence of PRJs.



(a) (b)

Figure 4. **(a) Comparison of RT among the downgraded GDS, other traditional algorithms and an M/M/1 queuing model; *PLOAD*=40%, the average computational volume of tasks is 8, the computer weight is 1.0 (b) Comparison between the GDS and the ideal bound (no PRJ); RF policy *MAX_CV*/*MIN_CV*=12/4**

As can be observed from Fig.4.a, the GDS outperforms other algorithms and shows the same performance as the M/M/1 queuing model except the arrival rate λ is greater than 0.08. The results coincide with the conclusion in Section 5, i.e. our algorithm is optimal in exploiting spare capabilities in a computer. Fig.4.b displays under the Response-First policy, the RT of *parallel real-time jobs* as a function of λ in a heterogeneous cluster of 8 computers under different levels of *PLOAD*. An ideal bound of RT is generated for comparison by running the same ARJ workloads in the same heterogeneous cluster in the absence of PRJs. The GR of the ARJs is also fixed to be 1.0. It is observed from Fig.4.b that in the case of 10% *PLOAD*, the RT obtained by the GDS is very close to the ideal bound, indicating the excellent performance of the GDS in utilizing spare capabilities to schedule parallel real-time jobs in a cluster scenario.



(a) (b)

Figure 5. **Effect of job workloads and the second-level selection policies on (a) GR (b) SU; Legends for Fig.5.a are the same as those in Fig.5.b**

Fig.5.a and b display the metrics GR and SU as the function of λ under the second-level selection policies of the Response-First and the Utilization-First, respectively. The first observation from Fig.5.a is that the GR decreases as λ increases in all cases, as expected. A further observation is that in the case of 10% and 40% *PLOAD*, the RF policy outperforms the UF when λ is low, while when λ exceeds a threshold, the opposite is true. This may be because that the RF policy is predisposed to first choose the better computers, whereas UF has the opposite trait. The experimental results suggest that when the ARJ workload is so high that all the computers in a heterogeneous cluster will be heavily occupied, allocating workload to poorer nodes and then to better nodes is more appropriate than allocating in the opposite direction.

As can be observed from Fig.5.b, SU increases as λ increases, as is to be expected. It is also observed that the UF policy outperforms RF (in terms of SU) in cases of 10% and 40% *PLOAD*, especially when λ is low. This can be explained as follows. Allocating a task on the poorer computer will lead to more of an increase in SU than allocating it on the better computer, if the makespan of the current workloads remains unchanged or increases just by a small amount. However, if a task allocation will cause a big increase in the makespan, the allocation may cause a decrease in SU. Fortunately, such an allocation has less chance of passing the admission control, since it implies a very late finish of the task. Hence, through the filtering of the admission control, the UF policy contrib-

utes more to the improvement of SU than the RF policy. The experimental results suggest that utilization of the cluster is significantly enhanced compared with the original *PLOAD*.

6.2. Computation and communication heterogeneity

Fig.6.a shows the impact of *computational heterogeneity* on the metrics GR and SU. Fig.6.b and Fig.6.c show the impact of *communicational heterogeneity* on GR and SU, respectively, under different levels of the computational heterogeneity. Only the results for 40% *PLOAD* and the RF policy are shown as the results for the other cases demonstrate similar patterns.

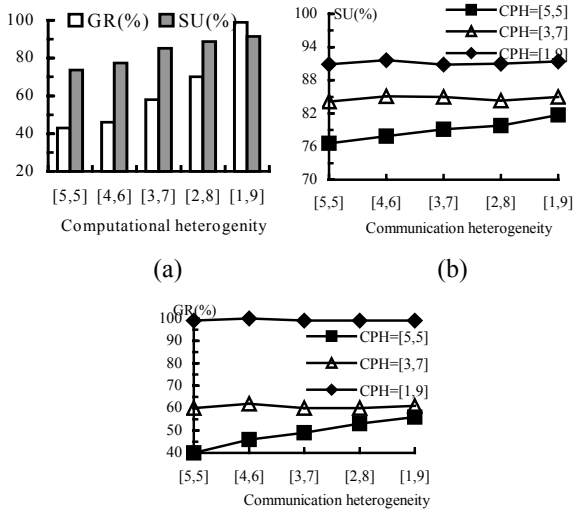


Figure 6. **Effect of computation heterogeneity and communication heterogeneity, the RF policy and *PLOAD*=40% (a) Effect of computational heterogeneity on GR and SU, λ=0.006 (b) Effect of the communication heterogeneity on SU, λ=0.006 (c) Effect of the communication heterogeneity on GR, λ=0.006**

The levels of computation and communication heterogeneity are measured by the scale of the range from which computer weights and communication link weights are selected. Five sets of computer and link weights, all with the same average, are uniformly chosen from five ranges, [1,9], [2,8], [3,7], [4,6] and [5,5].

As can be observed from Fig.6.a, GR and SU improve as the computational heterogeneity increases. The increase in GR may be because as the computation heterogeneity increases, the increasing variance in a task's execution time provides the task with more chance of fitting into the idle time slots before its deadline in the cluster. Under the same workloads, the increase in GR leads to an increase in SU. It can be observed from Fig.6.b and Fig.6.c that SU and GR increase as the communication heterogeneity increases in the case when the computation heterogeneity is [5,5] (i.e. homogeneity). This may be because the increasing variance in message transfer time provides more chance of finding a suitable idle time slot in the communication channels. However, the communication heterogeneity has no obvious impact on SU and GR when the computation heterogeneity increases to [3,7] or [1,9]. This suggests that the level of computation heterogeneity is more critical for scheduling ARJs than the level of the communication heterogeneity.

6.3 Task size and message size

Fig.7.a shows the impact of the size of tasks in ARJs on GR and SU. Only the results for 40% *PLOAD* are shown; the results for the other levels of *PLOAD* show similar patterns. The task size is measured by the average computational volume of tasks in an ARJ. In this experiment, when the task size increases, the average arrival rate λ is set to decrease proportionally so as to keep the total ARJ workload unchanged.

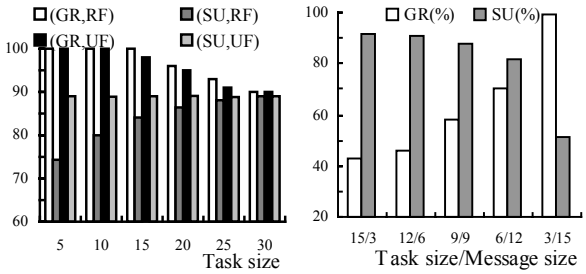


Figure 7. **Effect of task and message size, *PLOAD*=40% (a) Effect of task size on GR and SU, λ is 0.025 when the task size is 5 (b) Effect of task-size/message-size ratio on GR and SU, λ=0.02, RF**

As can be observed from Fig.7.a, under RF policy, the impact of the task size is mixed. On the one hand, GR retains 100% and then decreases as the task size increases. This is because the longer a task is run, the more chance there is that the task is disrupted by PRJs. It can be concluded from this result that under the same workloads, this admission control will favour dense, short new jobs. On the other hand, SU improves as the task size increases. This is because under the RF policy, a smaller GR may be an implication that many tasks are allocated to poorer computers, which leads to an improvement of SU. Under the UF policy the GR decreases, but the SU remains stable as the task size increases. This may be because the UF policy allocates tasks first on poorer computers, so the decline in GR has no obvious impact on SU.

Fig.7.b demonstrates the effect, on GR and SU, of the ratio of the task size to the size of messages among tasks. Only the results for 40% *PLOAD* and the RF policy are presented since other cases have similar behaviours. The message size of an ARJ is measured by the average volume of all messages among tasks in the ARJ. The task-size/message-size ratio varies from 15/3 to 3/15, all with the same volume sum. As can be observed from Fig.7.b, the impact of the task-size/message-size ratio is also mixed. GR improves but SU decreases as the task-size/message-size ratio increases. The increase in GR can be explained as follows. First, it is easier for small tasks to be admitted as demonstrated above. Second, although the message size increases as the task-size/message-size ratio decreases, the scheduling policy may compensate for this by scheduling two tasks on the same computer if the communication time between them is too long. Finally, computers are shared by ARJs and PRJs, while communication links are exclusively utilized by ARJs. These results indicate that in an ARJ, the task size is a more critical parameter than the message size for the ARJ's admission and response time.

7. Conclusions

In this paper a scheduling framework is presented to schedule dynamic aperiodic parallel real time jobs by modelling spare capabilities of a heterogeneous cluster on which periodic real-time jobs are running. The approach of spare-capability modelling is optimal in the sense that once a new task starts running, it will utilize all spare capability and its finish time is the earliest possible. No communication overheads are incurred by this approach. Scheduling for ARJs takes both task and message scheduling into account. Extensive simulations are conducted that show that system utilization is significantly enhanced without impacting on the QoS of existing jobs. Future studies are planed to extend the scheduling scheme to take the prediction, scheduling and dispatch time of ARJs into account.

8. References

[1] M. Apte, S. Chakravarthi, J. Padmanabhan and A. Skiellum, "A Synchronized Real-Time Linux Based Myrinet Cluster for Deterministic High Performance Computing and MPI/RT," *The 15th International Parallel and Distributed Processing Symposium*, 2001.

[2] T. F. Abdelzaher, K. G. Shin, "Combined Task and Message Scheduling in Distributed Real-Time Systems," *IEEE Transactions on parallel and distributed systems*, 10(11), November 1999.

[3] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Optimal reward-based scheduling of periodic real-time tasks," *The 20th IEEE Real-Time Systems Symposium*, December 1999.

[4] M. Caccamo, G. Lipari, and G. Buttazzo, "Sharing Resources with the TB* Server," *IEEE Real-Time Systems Symposium*, 1999.

[5] D. B. Golub, "Operating System Support for Coexistence of Real-Time and Conventional Scheduling," *The 1st Symposium on Operating Systems Design and Implementation*, 1994.

[6] K. Hwang and Z. Xu, Scalable Parallel Computing: Technology, Architecture, Programming. McGraw Hill, 1998.

[7] D. Kebbal, E.G. Talbi, J.M. Geib, "Building and Scheduling Parallel Adaptive Applications in Heterogeneous Environments," *1st IEEE Computer Society International Workshop on Cluster Computing*, December, 1999.

[8] J. P. Lehoczky and S. Ramos-Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems," *Proc. of Real-Time Systems Symposium*, 1992, pp.110-123.

[9] C.W. Mercer, S. Savage, and H. Tokuda, "Processor Capacity Reserves: Operating System Support for Multimedia Applications," Proc of the IEEE International Conference on Multimedia Computing and Systems, 1994.

[10] G.R. Nudd, D.J.Kerbyson et al, "PACE-a toolset for the performance prediction of parallel and distributed systems," *Intl Journal of High Performance Computing Applications, Special Issues on Performance Modelling*, 14(3), 2000, 228-251.

[11] X. Qin and H. Jiang, "Dynamic, Reliability-driven Scheduling of Parallel Real-time Jobs in Heterogeneous Systems," *The 30th International Conference on Parallel Processing*, Valencia, Spain, September 3-7, 2001.

[12] S. Ranaweera, and D. P. Agrawal, "Scheduling of Periodic Time Critical Applications for Pipelined Execution on Heterogeneous Systems," *Proceedings of the 2001 International Conference on Parallel Processing*, 2001.

[13] D A. E Salaheddine, "Aperiodic Scheduling in a Dynamic Real-Time Manufacturing System," *IEEE Real-Time Embedded System Workshop*, 2001.

[14] D. P. Spooner, S. A. Jarvis, J. Cao, S. Saini and GR. Nudd, "Local Grid Scheduling Techniques using Performance Prediction," *IEE Proc-Computers and Digital Techniques*, 150(2): 87-96, 2003.

[15] M. Spuri and G. Buttazzo, "Scheduling Aperiodic Tasks in Dynamic Priority Systems," *Real-Time Systems* 10(2), 1996, 179-210.

[16] B. Srinivasan, S. Pather, F. Ansari, and D. Niehaus, "A Firm Real-Time System Implementation Using Commercial Off-The-Shelf Hardware and Free Software," *The 4th IEEE Real-Time Technology and Applications Symposium*, 1998

[17] X.Y. Tang, S.T. Chanson, "Optimizing static job scheduling in a network of heterogeneous computers," *29th Intl Conference on Parallel Processing*, 2000.

[18] M. Thomadakis and J. Liu, "On the Efficient Scheduling of Non-periodic Tasks in Hard Real-Time Systems," *IEEE Real-time System Symposium*, 1999.

[19] H. Topcuoglu, S. Hariri and M. Wu, "Task Scheduling Algorithms for Heterogeneous Processors," *The Eighth Heterogeneous Computing Workshop*, 1999

[20] M. Wu, W. Shu and Y. Chen, "Runtime Parallel Incremental Scheduling of DAGs," *International Conference on Parallel Processing*, 2000.