Dynamic, Capability-driven Scheduling of DAG-based Real-time Jobs in Heterogeneous Clusters

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

***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 each parallel real-time job 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 will be the earliest possible. The performance of the proposed modelling and scheduling is evaluated through extensive simulation; the results show that the system utilization is significantly enhanced, while the real-time requirements of the existing jobs remain guaranteed.**

***Index Terms*—cluster computing, dynamic scheduling, spare capabilities, heterogeneous clusters, periodic real-time jobs, DAG real-time jobs, and performance prediction.**

I. Introduction

Cluster systems are gaining in popularity for the processing of scientific and commercial applications [9]. The research has also taken place to extend conventional operating systems (such as Linux) to support real-time scheduling (e.g. preemptive scheduling using the Earliest-Deadline-First policy) [7], [23], and as a result cluster systems are increasingly used for the processing of applications with time constraints [1], [24]. Many scenarios about real-time processing can be represented abstractly, as the hybrid execution of both existing periodic jobs and 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 some required level [13]. These reserved executions can be viewed as periodic jobs and in addition to these the underlying processors also have 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 further complicated by trying 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) [15]. In this scheduling framework, a global scheduler - located on a computer in the heterogeneous cluster - analyzes the execution of PRJs on the remaining computers and models the initial distribution of their spare capabilities off-line. Once a new ARJ arrives at the system, the global scheduler releases the precedence constraints among the tasks in the ARJ, adjusts the initial distribution of spare capabilities (as this may have changed due to the execution of preceding ARJs) and then tries to place the execution of the tasks from the new ARJ into the spare time slots that are available. The response times of the tasks in the ARJ are computed, to determine whether the job can be accepted. If this is the case then the tasks in the ARJ will be sent to the designated computers and executed in parallel, exploiting the spare capabilities. Global scheduling for ARJs takes both task and message scheduling into account and local scheduling, at each computer, uses a uniform Early Deadline First (EDF) scheduling policy.

The modelling of spare capabilities proposed in this paper does not invoke any additional communication 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 on a computer, it will utilize all the spare capability left by the PRJs, and its finish time is the earliest possible. This work is supported by an existing Performance Analysis and

Manuscript received Dec 31, 2003. 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).

The authors are with the Department of Computer Science, University of Warwick, Coventry, CV4 7AL, United Kingdom. E-mail: {liganghe, saj, dps, grn}@dcs.warwick.ac.uk.

Characterization Environment (PACE) [14]. PACE can be used to predict application behavior and provide performance data (such as the execution time of a job) which supports the task of scheduling and resource management [6], [21].

The remainder of this paper is organized as follows: Section II presents the related work; Section III describes the workload and system model; in Section IV a novel approach is presented that allows a global scheduler to model the spare capabilities of computers in a cluster; Section V describes a global dynamic scheduling algorithm for DAG real-time jobs; a performance evaluation of this modelling approach and scheduling algorithm is presented in Section VI and Section VII concludes the paper.

II. RELATED WORK

The study of heterogeneous clusters or networks of workstations has received a good deal attention [10], [11], [20], [25]. The scheduling of tasks with precedence constraints, which are usually represented by *Directed Acyclic Graphs* (DAG), has also been well documented [2], [11], [15], [16], [26], [28], [30]. 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. In [30] a dynamic incremental DAG scheduling approach for parallel machines is described. However, the approach is limited to homogenous systems. Two low-complexity efficient heuristics, the Heterogeneous Earliest-Finish-Time Algorithm and the Critical-Path-on-a-Processor Algorithm are proposed in [28], each for the scheduling of DAGs on heterogeneous processors. These heuristics are not designed for real-time task allocation and are therefore not suitable for scheduling in a real-time context because job requirements cannot be guaranteed. In [15] non-real-time DAGs are extended to include real-time information, and the scheduling of parallel tasks with real-time DAG topologies on heterogeneous systems is proposed. This technique differs from that presented in this paper because it is not aimed at the utilization of spare system capabilities.

A number of scheduling algorithms for periodic real-time jobs on multi-computer or multiprocessor systems have also been presented [3], [4], [17], [18]. A task duplication technique combined with pipelined execution for the scheduling of time critical periodic applications on heterogeneous systems can be found in [17]. In [3], a reward-based scheduling scheme for periodic tasks is presented. While these techniques are effective, they are unable to deal with the hybrid execution of periodic and aperiodic tasks.

Scheduling systems for the processing of both periodic and aperiodic real-time tasks can be classified into fixed or dynamic priority systems. Fixed priority systems assume that the priority of each periodic task is fixed, whereas in dynamic priority systems different instances of periodic tasks may have different priorities (such as in scheduling systems that use EDF). Dynamic priority systems typically attain higher processor utilization than fixed ones. Slack Stealing policies have been designed for fixed priority systems [12], 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 [5], [19], [22], [29]. 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 to be independent and without precedence constraints. Techniques presented in [12] and [27] run aperiodic tasks by using the spare capabilities left by periodic tasks; though in each case they are limited to uniprocessor scenarios. [8] presents an algorithm to jointly schedule both periodic and aperiodic tasks on clusters. This however is not aimed at the exploitation of spare capabilities. In this same paper it is also assumed that aperiodic tasks are independent and non-real-time, that the cluster is homogeneous and that all periodic tasks start at the same time.

Extending the modelling of spare capabilities from uniprocessor architectures to cluster environments is non-trivial. This is primarily because a uniprocessor system only needs to model the spare capabilities within itself, whereas in a cluster a central node models the spare capabilities of the other nodes in the system, making the information needed for this calculation far more difficult to attain. The modelling approach in this paper efficiently models the spare capabilities of computers in a heterogeneous cluster and significantly, is free of additional communication overheads. The proposed scheme is also designed for dynamic priority systems where an EDF policy is used for local scheduling.

III. WORKLOAD AND SYSTEM MODEL

A heterogeneous cluster of computers is modelled 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 message unit 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 using the triple (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 (times are measured in *time units* unless otherwise stated). 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.

ARJs arrive at the heterogeneous cluster dynamically. If

accepted, an ARJ is run once. An ARJ is modelled 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 the *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 states that v_j will start to run 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 e_{ij}'s message volume.

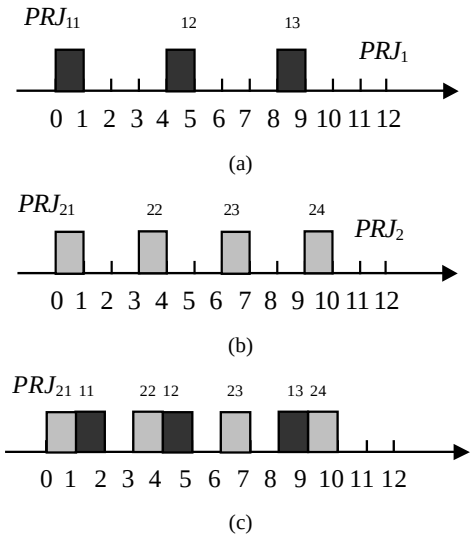


Fig. 1. A case study of PRJs (a) PRJ_1 with a period of 4 and an execution time of 1, (b) PRJ_2 with a period of 3 and an execution time of 1, (c) execution of PRJ_1 and PRJ_2 under EDF starting at 0

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

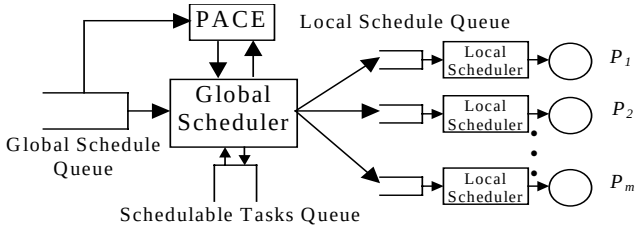


Fig. 2. The scheduler model in the heterogeneous cluster environment

All ARJs arrive at the global scheduler where they wait in a *global schedule queue*. These new jobs are serviced on a First-Come-First-Served basis. Each time the global scheduler fetches a job from the head of the global schedule queue, it searches for schedulable tasks in the ARJ and inserts them into the schedulable task queue so that the deadlines of the tasks in the schedulable task queue are in increasing order.

The global scheduler then picks a task from the head of the schedulable task queue and schedules it globally. A task in an ARJ is considered schedulable if either the task has no predecessors or all of its predecessors have been scheduled. A task is considered acceptable by a computer if it can be completed before its deadline and the real-time requirements of all the PRJs on that computer remain guaranteed. If a task is not acceptable by any of the computers then it is rejected. Consequently, the ARJ that the task belongs to is rejected. When the global scheduler finishes scheduling a task, it searches for new schedulable tasks in the ARJ and updates the schedulable task queue. An ARJ is accepted if all its composite tasks are acceptable.

Once accepted, the tasks in the ARJ are sent to the local schedulers of the designated computers. At each computer, the local scheduler receives the new tasks and inserts them into a *local schedule queue*, ensuring that the deadlines of the tasks (ARJ's tasks or the PRJs' PJIs) are in increasing order. The local scheduler uniformly schedules both the ARJs' tasks and the PRJs' PJIs using EDF. Each task to be executed is fetched from the head of the local schedule queue and the local schedule is preemptive.

In this scheduler model, PACE accepts ARJs, predicts the execution time of each task in the ARJs on each computer and then 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. In the current model it is assumed that the execution and communication times predicted by PACE are entirely accurate, and that the times for prediction, scheduling and task dispatching are negligible (extending the model is the subject of future work).

IV. SPARE CAPABILITY MODELLING

In this section, the initial distribution of idle time left by the PRJs (i.e. the spare capabilities left by the PRJs before certain time points) is modelled. This modelling procedure can be done off-line. 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 new arriving ARJ.

A. Off-line Modelling of the Initial Distribution of the Spare Capability

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.

The above calculation can be performed for all *PJI*s of any PRJ_i running on the same computer; 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) and P_{ij}

is the sum of the execution times of *PJI*s that must be complete before D_{ij}.

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

P_{ij} can be calculated as Eq.2, where S_k is PRJ_k's start time.

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

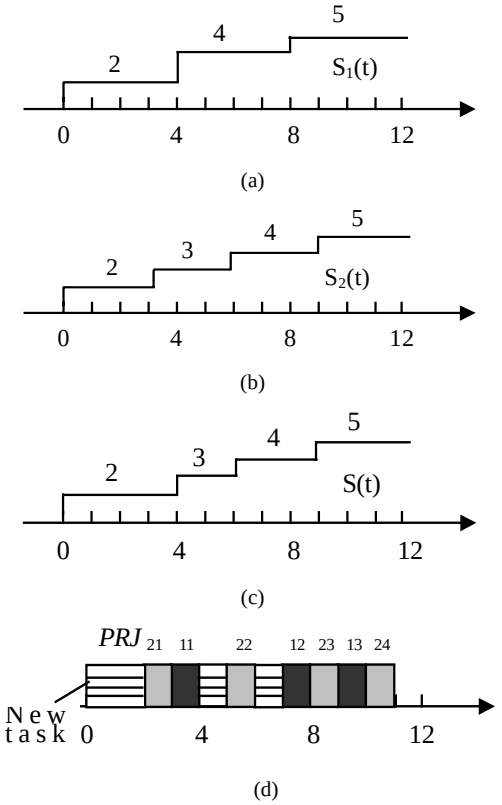


Fig. 3. A case study of the function of idle time units (a) Function of idle time units for PRJ1 (b) Function of idle time units for PRJ2 (c) Function of idle time units for both PRJ1 and PRJ2 (d) The joint execution of PRJ1, PRJ2 and a new task with an execution time of 4 starting at 0

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* are 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 PRJ set, 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.

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

B. 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 begin before time t_0 but must also be complete before the *JTP*. Some additional 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.

$PJ(t_0) = \{PJI_{ij} | D_{ij} \le t_0\}$ (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 using 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 let JTP_1 be 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 $LJ_k(t_0)$.

$LJ_k(t_0) = \{ PJI_{ij} \mid R_{ij} < t_0 < D_{ij} \text{ and } D_{ij} \le JTP_k\}$ (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 the last subsection.

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:

$S(t_0, JTP_k) = S(JTP_k) - t_0 + P(t_0) + L_k(t_0)$ (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–$P(JTP_k)$=$S(JTP_k)$. Therefore Eq.7 holds. □

$L_k(t_0)$ in Eq.7 still remains unknown; the remainder of this subsection dedicated to this calculation.

If new tasks are run before t_0, the execution of PJIs in $PJ(t_0)$ may change so that they no longer retain the original execution pattern. Theorem 2 is introduced to reveal the distribution property of the remaining time units before t_0 after running the PJIs of $PJ(t_0)$ as well as any new tasks.

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

1) either the 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, or

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.

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

Proof: The execution of new tasks may delay the execution of PJIs in $PJ(t_0)$. The delayed PJIs may also delay other PJIs in $PJ(t_0)$ still further. This chain of delays 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, only PRJ_1 and PRJ_2 are run. In Fig.3.d, the current last executed new task finishes at time 7. 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. t_s can be set to time 11 in the example. There are no idle time slots between time 7 and 11 (time 7 is the end-time of the new task).

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. This is shown in Algorithm 1, where $I_P(s, t_0)$ is the number of time units that PJIs in $LJ_k(t_0)$ can occupy in time period $[s, t_0]$.

Algorithm 1. Calculating $L_k(t_0)$

1. L←$LJ_k(t_0)$; $L_k(t_0)$←0;
2. **while**(L≠Φ)
3. LI←the PJI with the least deadline in L;
4. c← LI's execution time; s← LI's ready time;
5. uf←the unfinished workload of LI;
6. **if** (s<t_s) **then** s←t_s;
7. **if** ($I_P(s, t_0)$>uf) **then**
8. The finished workload of LI before t_0 is c;
9. $L_k(t_0)$←$L_k(t_0)$+c;
10. Deduct the time units used to run LI in $[s, t_0]$ from $I_P(t_s, t_0)$;
11. **else** LI's finished workload in $[0, t_0]$is c-uf+$I_P(s, t_0)$; $L_k(t_0)$←$L_k(t_0)$+c-uf+$I_P(s, t_0)$;
12. Deduct the time units used to run LI in $[s, t_0]$ from $I_P(t_s, t_0)$;
13. L←L–LI;
14. **end while**

V. SCHEDULING ALGORITHMS

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.

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

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≠k) and there exists a message schedule sequence, (mst_1^{sk}, mft_1^{sk}), $(mst_2^{sk}, mft_2^{sk}),\ldots, (mst_a^{sk}, mft_a^{sk})$, 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, respectively; 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 $(mft_b^{sk}, mst_{b+1}^{sk})$.

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

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

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

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

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

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 into the schedulable task queue. Then, at each step, the global scheduler picks a task from the head of the schedulable task 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 2.

Algorithm 2 Calculating the finish time of task v_i starting at t_0 on 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 3.

Algorithm 3 Admission Control for task v_i

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 2 to calculate v_i's finish time on p_j, $ft^j(v_i)$;
5. **if** $(ft^j(v_i) \le dt(v_i))$ **then**
6. PC=PC$\cup\{p_j\}$;
7. **end for**
8. **if** PC$=\Phi$ **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 on more than one computer, two possible *Second-level Selection Policies* are offered to choose a final computer. The first is a *Response First* (RF) policy, which selects the computer on which v_i has the earliest finish time. The second is a *Utilization First* (UF) policy, which selects the computer on which v_i has the longest execution time. The two policies are motivated in different ways. The RF policy will select computers with a better performance, so that tasks can be completed sooner; the UF policy on the other hand aims to improve utilization by increasing the chance of selecting poorer performing computers. In section VI the performance of these two policies is evaluated.

After deciding which computer 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 then sent to the designated computers.

When the local scheduler at some computer receives the allocated ARJs' tasks, or the PJIs of the PRJs are ready for execution, it inserts these into the *Local Schedule Queue* ordered by increasing deadlines. Execution proceeds from the head of the queue and once the task with the lowest deadline is ready, the current execution is preempted.

Assuming 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 4.

Algorithm 4 Global dynamic scheduling for parallel real-time jobs

1. **if** global scheduler queue$=\Phi$ **then** wait until a new ARJ arrives, then go to step 3;
2. **else**
3. Get a job from the head of the 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 3 to judge whether to accept or reject v_i;
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 the 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 IV, v_i cannot be finished earlier on the computer on which the new task is scheduled, otherwise the deadlines of some of the PJIs on that computer will be missed. In this sense, the modelling approach is optimal.

When the global scheduler models the spare capacities of other computers in the cluster, no additional information has to be transferred among them to allow a scheduling decision to be made. Hence no communication overhead is incurred by the modelling approach.

VI. PERFORMANCE EVALUATION

The experimental parameters in this simulation are chosen either based on those used in the literature [12], [15] 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 the 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 communicational 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 I
PARAMETERS FOR SIMULATION STUDIES

Parameter	Definition	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	The number of computers 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 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 time spent running tasks to the total available time 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.

A. Job Workloads and Second-level Selection Policies

The RT can be viewed as a measure of the ability of the scheme to utilize the spare capabilities in the component 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 [5], [19], [22], [29]; 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 on a cluster, whereas the 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 and the other housing a local scheduler and jointly running tasks; the computational volumes of tasks follow an exponential distribution. To stress the response performance, GR of non-periodic real-time tasks is fixed at 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 in the absence of PRJs.

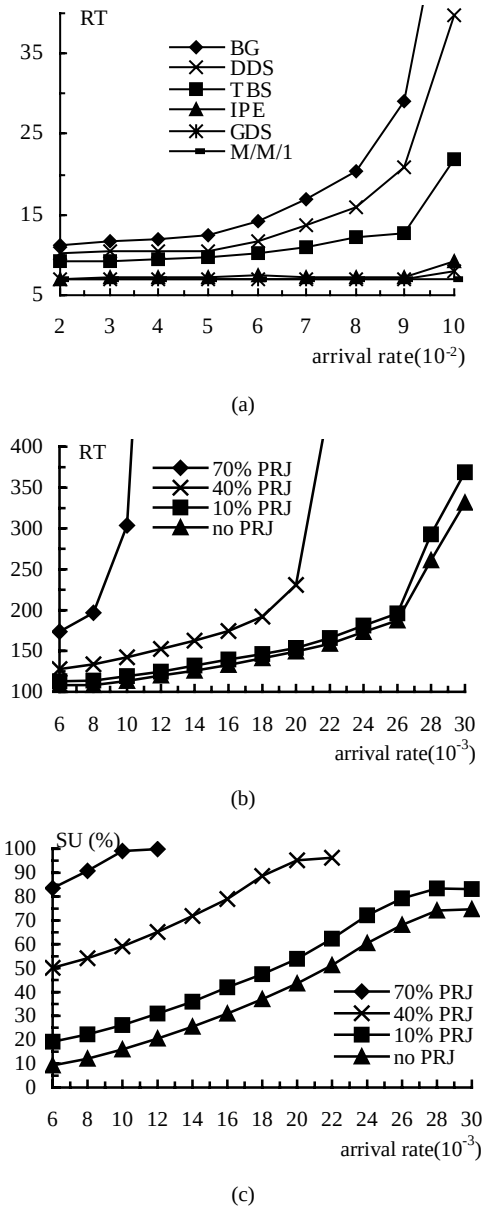


Fig. 4. (a) Comparison of RT among the downgraded GDS, other traditional algorithms and an M/M/1 queuing model; PLOAD=40%, the average computational volume of tasks is 8 and the computer weight is 1.0 (b) Comparison between the GDS and the ideal bound (no PRJ); MAX_CV/MIN_CV=12/4, RF policy (c) The corresponding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

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 V, 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 any 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 scheme is very close to the ideal bound, indicating the excellent performance of GDS in utilizing spare capabilities in the scheduling of parallel real-time jobs on a cluster. We also observe that in the case of 40% or 70% *PLOAD*, RT increases dramatically once λ is greater than some critical value. This is because the system utilization reaches a maximum value at that arrival rate. At 70% *PLOAD*, the maximum system utilization is near 100%, while at 40% the maximum is 97% (under the workload parameters selected for this experiment), this is shown in Fig.4.c. It is also observed from Fig.4.c that the maximum system utilization that can be reached under pure ARJ workloads is about 75%. The reason that the maximum system utilization is less than 100% is simply because of the existence of message passing and precedence constraints among the tasks in the ARJs.

As stated in Section V, when the deadline of a task in an ARJ can be met on more than one computer, the *Response First* (RF) policy or *Utilization First* (UF) policy are offered to aid the final selection. Fig.5.a, b and c display the metrics GR, SU and RT as the function of λ under these two policies, respectively. The first observation from Fig.5.a is that 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 UF when λ is low, whereas when λ exceeds a certain threshold the opposite is true. This is explained as follows. The RF policy is predisposed to choosing the better computers, whereas the UF policy has the opposite trait. When the ARJ workload is light, that is, not all computers are busy, allocating tasks to better computers will mean that there is more chance that the whole ARJ is accepted. By increasing the value of λ, we simulate all computers receiving new heavy workloads under both policies. In this context, under the UF policy and when tasks cannot be assigned to poorer computers, they have a greater chance of being allocated to better computers in order to meet their deadlines, compared with that of the RF policy which has the opposite selection preference. The experimental results suggest that when the ARJ workload is so high that all the computers in a heterogeneous cluster are heavily occupied, allocating workload to poorer nodes as opposed to better nodes is highly desirable.

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 the RF policy in terms of SU in cases of 10% and 40% *PLOAD*, especially when λ is low. This is because allocating a task to a poorer computer will lead to a greater increase in SU, if the makespan of the new task workload in the cluster remain unchanged (or increase by some small percentage). If the allocation of a task incurs a large increase in makespan however, the allocation may decrease the SU. Fortunately, such an allocation has less chance of passing the admission control since it implies a very late finish for the task. Hence, through the filtering function provided by the admission control, the UF policy contributes more to the improvement of SU than does the RF policy. The experimental results suggest that utilization of the cluster is significantly enhanced compared with the original *PLOAD*.

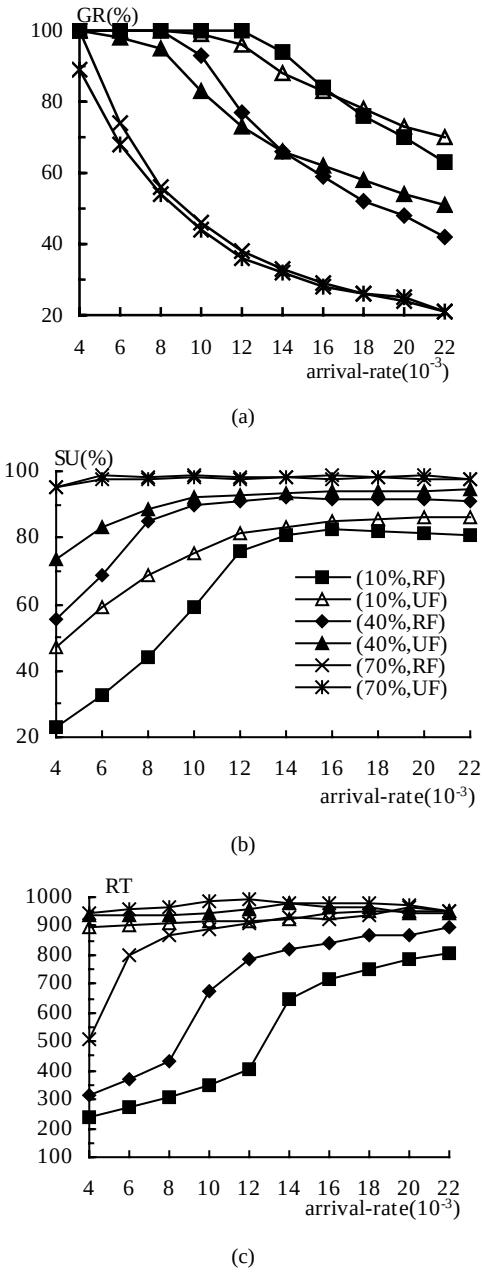


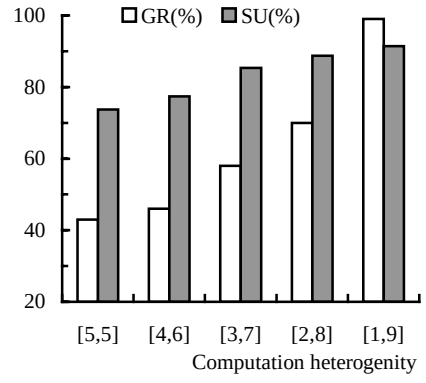
Fig. 5. Effect of Job workloads and second-level selection policy on (a) GR (b) SU and (c) RT, Legends for Fig.5.a and c are the same as those in Fig.5.b

It can be seen from Fig.5.c that under the RF policy the RT increases as λ increases and that the RT is greater under the UF policy than under the RF policy, especially when λ is low. This is caused by the selection predisposition of the UF policy. The result suggests that the UF policy has poorer performance than the RF policy in terms of RT. As can be observed from the three figures, in the case of 70% *PLOAD* the RF and UF policies have comparable performance. This is because when the

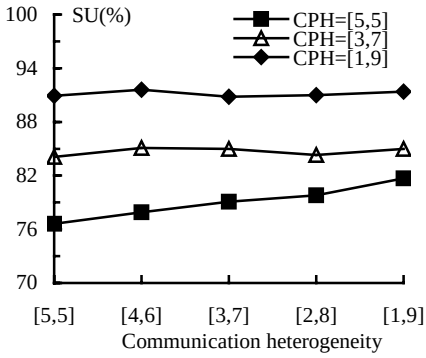
PLOAD in the cluster is high, there are a limited number of computers that satisfy the admission control.

B. Computation and Communication Heterogeneity

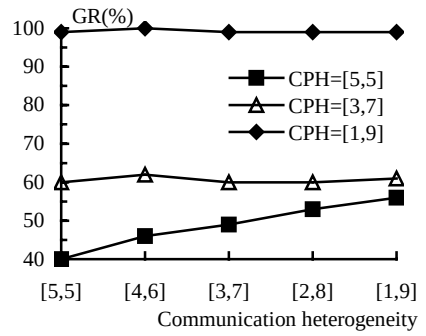
Fig.6.a shows the impact of *computation heterogeneity* on the metrics GR and SU. Fig.6.b and Fig.6.c show the impact of *communication heterogeneity* on GR and SU, respectively, under different levels of computation heterogeneity. Only the results for 40% *PLOAD* and the RF policy are shown, the results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



(a)



(b)



(c)

Fig. 6. Effect of computation and communication heterogeneity, the RF policy and PLOAD=40% (a) Effect of computation 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 computation 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. Under the same workload, 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 communication heterogeneity. The results suggest that the performance of the constituent computers in the cluster should be carefully selected so as to achieve balanced system performance.

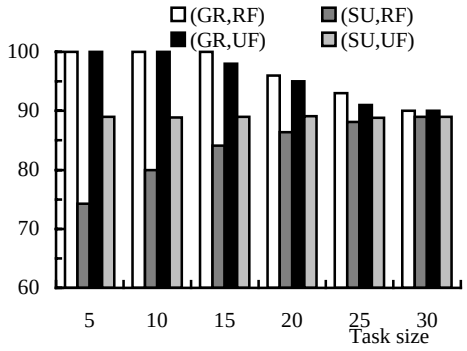
C. Task Size and Message Size

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 as the results for other levels of *PLOAD* display similar behaviour. 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.

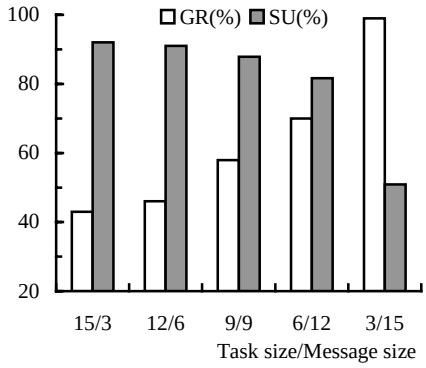
As can be observed from Fig.7.a, under the RF policy the impact of the task size is mixed. On the one hand, GR retains 100% and then decreases as the task size increases. This is because the longer a task is run, the more chance there is that the task is disrupted by the PRJs. It can be concluded from this result that under the same workload 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 lower GR may imply that a large number of tasks are being allocated to poorer computers, which contributes to an improvement in SU. Under the UF policy, GR decreases, but SU remains stable as the task size increases. This may be because the UF policy allocates tasks to poorer computers, so the decline in GR has no obvious impact on SU.

Fig.7.b and Fig.7.c demonstrate the effect of the ratio of the task size to the size of messages among tasks on GR, SU and RT. Again, only the results for 40% *PLOAD* and the RF policy are presented. The message size of an ARJ is measured by the average volume of all messages among the 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. The GR improves whereas the SU decreases as the *Task-size/message-size ratio* increases. The increase in GR can also be explained: First, it is easier for small tasks to be ad-

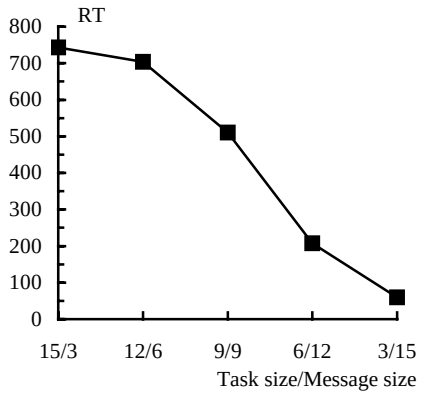
mitted as demonstrated above; second, although the message size increases as the *Task-size/message-size ratio* decreases, the scheduling policy may compensate for this by scheduling two tasks on the same computer if the communication time between them is too large; and finally, computers are shared by ARJs and PRJs, while the communication links are exclusively utilized by ARJs. Fig.7.c shows that the RT decreases as the *Task-size/message-size ratio* decreases. These results suggest that the task size is more critical than the message size when guaranteeing the ARJ's admission and response time.



(a)



(b)



(c)

Fig. 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 (c) Effect of task-size/message-size ratio on RT, with the same parameters as those in Fig.7.b

D. Cluster Size

The cluster size is changed in such a way as to keep the PRJ and PLJ workload unchanged while gradually removing the poorer computers from the cluster. As can be observed from Fig.8, the GR retains 100% when the number of computers in the cluster decreases from 16 to 12, and decreases when the cluster size decreases further from 12 to 6, while the SU improves as the cluster size decreases, as expected. Fig.8 suggests that a cluster with more than 12 computers is inefficient. This result indicates that the cluster size should be carefully selected based on experimental evidence so that a good cost-performance ratio can be achieved.

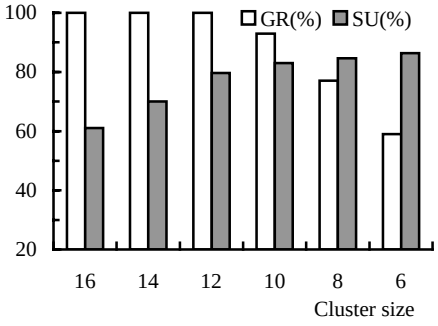


Fig. 8. Effect of the cluster size on GR and SU, λ=0.02, PLOAD is 10% in each computer as the number of computers is 16

VII. CONCLUSIONS

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

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," *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," *20th IEEE Real-Time Systems Symposium*, December 1999.

[4] M.A. Bonuccelli, M. C Clò, "EDD algorithm performance guarantee for periodic hard-real-time scheduling in distributed systems," *15th International Parallel and Distributed Processing Symposium*, 1999.

[5] M. Caccamo, G. Lipari, and G. Buttazzo, "Sharing resources with the TB* server," *IEEE Real-Time Systems Symposium*, 1999.

[6] J. Cao, S.A. Jarvis, S. Saini, D.J. Kerbyson and G.R. Nudd, "ARMS: An agent-based resource management system for grid computing," *Scientific Programming* (Special issue on Grid computing), 10(2): 135-48, 2002.

[7] D.B. Golub, "Operating system support for coexistence of real-time and conventional scheduling," *The 1st Symposium on Operating Systems Design and Implementation*, 1994.
[8] L. He, H. Jin, Y. Chen and Z. Han, "Optimal scheduling of aperiodic jobs on cluster system," *7th International Conference on Parallel and Distributed Computing* (Euro-Par 2001), Manchester, UK, 2001, pp.764-772.
[9] K. Hwang and Z. Xu, *Scalable Parallel Computing: Technology, Architecture, Programming*. McGraw Hill, 1998.
[10] 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.
[11] Y. Kwok, "Parallel program execution on a heterogeneous PC cluster using Task Duplication," *9th Heterogeneous Computing Workshop*, 2000.
[12] 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.
[13] 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.
[14] 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 Issue on Performance Modelling*, 14(3), 2000, 228-251.
[15] X. Qin and H. Jiang, "Dynamic, reliability-driven scheduling of parallel real-time jobs in heterogeneous systems," *30th International Conference on Parallel Processing*, Valencia, Spain, September 3-7, 2001.
[16] A. Radulescu, A. Gemund, "A low-cost approach towards mixed task and data parallel scheduling," *International Conference on Parallel Processing*, 2001.
[17] S. Ranaweera, and D.P. Agrawal, "Scheduling of periodic time critical applications for pipelined execution on heterogeneous systems," *Proc. of the 2001 International Conference on Parallel Processing*, 2001.
[18] S. Ranaweera, D.P. Agrawal, "Task duplication based scheduling algorithm for heterogeneous systems," *International Parallel and Distributed Processing Symposium*, 2000.
[19] D.A.E. Salaheddine, "Aperiodic scheduling in a dynamic real-time manufacturing system," *IEEE Real-Time Embedded System Workshop*, 2001.
[20] S. Sinha and M. Parashar, "Adaptive runtime partitioning of AMR applications on heterogeneous clusters," *3rd IEEE Intl Conf on Cluster Computing*, 2001.
[21] 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.
[22] M. Spuri and G. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," *Real-Time Systems*, 10(2), 1996, 179-210.
[23] B. Srinivasan, S. Pather, F. Ansari, and D. Niehaus, "A firm real-time system implementation using commercial off-the-shelf hardware and free software," *4th IEEE Real-Time Technology and Applications Symposium*, 1998
[24] M. Suzuki, H. Kobayashi, N. Yamasaki, and Y. Anzai, "A task migration scheme for high performance real-time cluster system," *19th International Conference on Computers and Their Applications*, pp. 228-231, 2003
[25] X.Y. Tang, S.T. Chanson, "Optimizing static job scheduling in a network of heterogeneous computers," *29th Intl Conference on Parallel Processing*, 2000.
[26] K. Taura, A. Chien, "A heuristic algorithm for mapping communicating tasks on heterogeneous resources," *9th Heterogeneous Computing Workshop*, 2000.
[27] M. Thomadakis and J. Liu, "On the efficient scheduling of non-periodic tasks in hard real-time systems," *IEEE Real-time System Symposium*, 1999.
[28] H. Topcuoglu, S. Hariri and M. Wu, "Task scheduling algorithms for heterogeneous processors," *8th Heterogeneous Computing Workshop*, 1999.
[29] S. Wang, Y.C. Wang, K. Lin, "Integrating the fixed priority scheduling and the total bandwidth server for aperiodic tasks," *7th International Conference on Real-Time Systems and Applications*, 2000.
[30] M. Wu, W. Shu and Y. Chen, "Runtime parallel incremental scheduling of DAGs," *International Conference on Parallel Processing*, 2000.

Ligang He is a PhD student and research associate in the High Performance System Group in the Department of Computer Science at the University of Warwick, United Kingdom. His areas of interests are cluster computing, grid computing, real-time scheduling and performance prediction. He is a student member of the IEEE.

Dr. Stephen Jarvis is a Senior Lecturer in the High Performance System Group at the University of Warwick. He has authored over 50 referred publications (including three books) in the area of software and performance evaluation. While previously at the Oxford University Computing Laboratory, he worked on performance tools for a number of different programming paradigms including the Bulk Synchronous Parallel (BSP) programming library – with Oxford Parallel and Sychron Ltd – and the Glasgow Haskell Compiler – with Glasgow University and Microsoft Research in Cambridge. He has close research links with IBM, including current projects with IBM's TJ Watson Research Center in New York and with their development centre at Hursley Park in the UK. Dr Jarvis sits on a number of international programme committees for high-performance computing, autonomic computing and active middleware; he is also the Manager of the Midlands e-Science Technical Forum on Grid Technologies.

Daniel Spooner is a newly-appointed Lecturer in the Department of Computer Science and is a member of the High Performance System Group. He has 15 referred publications on the generation and application of analytical performance models to Grid computing systems. He has collaborated with NASA on the development of performance-aware schedulers and has extended these through the e-Science Programme for multi-domain resource management. He has worked at the Performance and Architectures Laboratory at the Los Alamos National Laboratory on performance tools development for ASCI applications and architectures.

Graham Nudd is head of the High Performance Computing Group and Chairman of the Computer Science Department at the University of Warwick. His primary research interests are in the management and application of distributed computing. Prior to joining Warwick in 1984, he was employed at the Hughes Research Laboratories in Malibu, California.