

# Performance-Based Dynamic Scheduling of Hybrid Real-Time Applications on a Cluster of Heterogeneous Workstations<sup>1</sup>

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

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

**Abstract.** It is assumed in this paper that periodic real-time applications are being run on a cluster of heterogeneous workstations, and new non-periodic real-time applications arrive at the system dynamically. In the dynamic scheduling scheme presented in this paper, the new applications are scheduled in such a way that they utilize spare capabilities left by existing periodic applications in the cluster. An admission control is introduced so that new applications are rejected by the system if their deadlines cannot be met. The effectiveness of the proposed scheduling scheme has been evaluated using simulation; experimental results show that the system utilization is significantly improved.

## 1 Introduction

In cluster environments the nodes are rarely fully utilized [1]. In order to make use of the spare computational resources, scheduling schemes are needed to judiciously deal with the hybrid execution of existing and newly arriving tasks [2]. The work in this paper addresses the issue. This work has two major contributions. First, an optimal approach for modeling the spare capabilities of clusters is presented. Second, based on the modeling approach, a dynamic scheduling framework is proposed to allocate newly arriving independent *non-periodic real-time applications (NPA)* to a heterogeneous cluster on which *periodic real-time applications (PRA)* are running.

## 2 System Modeling

A cluster of heterogeneous workstations is modeled as  $P = \{p_1, p_2, \dots, p_m\}$ , where  $p_i$  is an autonomous workstation [4]. Each workstation  $p_i$  is weighted  $w_i$ , which represents the time it takes to perform one unit of computation. Each workstation has a set of *PRAs*. On a workstation with  $n$  *PRAs*, the  $i$ -th periodic real-time application *PRA* <sub>$i$</sub>  ( $1 \leq i \leq n$ ) is defined as  $(S_i, C_i, T_i)$ , where  $S_i$  is the *PRA* <sub>$i$</sub> 's start time,  $C_i$  is its execution

---

<sup>1</sup> This work is sponsored in part by grants from the NASA AMES Research Center (administered 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).

time (in *time units*) on the workstation, and  $T_i$  is the  $PRA_i$ 's period. An execution of  $PRA_i$  is called a *periodic application instance (PAI)* and the  $j$ -th execution is denoted as  $PAI_{ij}$ .  $PAI_{ij}$  is ready at time  $(j-1)*T_i$ , termed the ready time ( $R_{ij}$ ,  $R_{ij}=S_i$ ), and must be complete before  $j*T_i$ , termed the deadline ( $D_{ij}$ ). All  $PAIs$  must meet their deadlines and are scheduled using an Early Deadline First (*EDF*) policy. The  $i$ -th arriving *NPA*,  $NPA_i$ , is modeled as  $(a_i, cv_i, d_i)$ , where  $a_i$  is  $NPA_i$ 's arrival time,  $cv_i$  is its computational volume and  $d_i$  is its deadline. The execution time of  $NPA_i$  on workstation  $k$  is denoted as  $c^k(cv_i)$ .

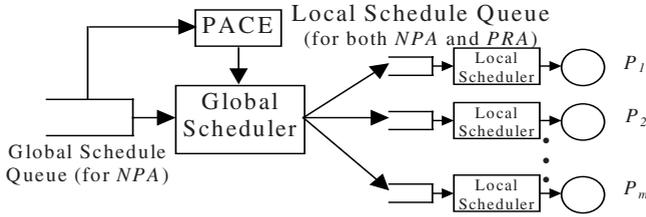


Fig. 1. The scheduler model in the heterogeneous cluster environment

Fig.1 depicts the scheduler model in the heterogeneous cluster. It is assumed that the  $PRAs$  are running in the workstations. All  $NPA$ s arrive at the global scheduler, where they wait in a *global schedule queue (GSQ)*. Each  $NPA$  from the  $GSQ$  is globally scheduled and, if accepted, sent to the *local scheduler* of the designated workstation. At each workstation, the *local scheduler* receives the new  $NPA$ s and inserts them into a *local schedule queue (LSQ)* in order of increasing deadlines. The local scheduler schedules both  $NPA$ s and  $PRA$ s'  $PAIs$  in the  $LSQ$  uniformly using  $EDF$ . The local schedule is preemptive. In this scheduler model, the  $PACE$  toolkit [3] accepts  $NPA$ s, predicts their execution time on each workstation in real-time and returns the predicted time to the global scheduler [5].

### 3 Scheduling Analysis

A function constructed of idle time units, denoted as  $S_i(t)$ , is defined in Equ.1.  $P_{ij}$  is the sum of execution time of the  $PAIs$  that must be completed before  $D_{ij}$ .  $P_{ij}$  can be calculated as Equ.2, where  $S_k$  is  $PRA_k$ 's start time.

$$S_i(t) = D_{ij} - P_{ij} \quad D_{i(j-1)} < t \leq D_{ij}, \quad 1 \leq i \leq n, \quad j \geq 1 \quad \text{where } D_{i0} = 0 \tag{1}$$

$$P_{ij} = \sum_{k=1}^n \left[ \alpha / T_k \right] * C_k, \quad \text{where, } \alpha = \begin{cases} D_{ij} - S_k & D_{ij} > S_k \\ 0 & D_{ij} \leq S_k \end{cases} \tag{2}$$

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

In the function  $S_i(t)$ , the time points, except zero, at which the function value increases, are called *Jumping Time Points (JTP)*. If the number of time units that are used to run  $NPA$ s between time 0 and any  $JTP$  is less than  $S_i(JTP)$ , the deadlines of all  $PAIs$  of  $PRA_i$  can be guaranteed. Suppose  $n$   $PRA$ s ( $PRA_1, \dots, PRA_r, \dots, PRA_n$ ) are running

on the workstation, then the function of idle time units, denoted as  $S(t)$ , for the  $PRA$  set can be derived from the individual  $S_i(t)$  ( $1 < i < n$ ), shown in Equ.3.  $S(t)$  suggests that idle time units of  $S(JTP)$  are available in  $[0, JTP]$ .

If a  $NPA$  arrives and starts running at any time  $t_0$ , the remaining idle time slots in  $[t_0, JTP]$ , denoted as  $S(t_0, JTP)$ , is calculated in Theorem 1. Some additional notation is introduced to facilitate the description.  $PA(t_0)$  is a set of  $PAIs$  whose deadlines are no more than time  $t_0$ , defined in Equ.4.  $LA(t_0)$  is a set of  $PAIs$  whose deadlines are more than  $t_0$ , but whose ready times are less than  $t_0$ , defined in Equ.5.  $P(t_0)$  and  $L(t_0)$  are the number of time units in  $[0, t_0]$  that are used for running the  $PAIs$  in  $PA(t_0)$  and  $LA(t_0)$ , respectively.  $P(t_0)$  can be calculated as Equ.6. Let  $JTP_1, JTP_2, \dots, JTP_k$  be a sequence of  $JTPs$  after  $t_0$  in  $S(t)$  and  $JTP_1$  the nearest to  $t_0$ .  $LA_k(t_0)$ , defined in Equ.7, is a set of  $PAIs$ , whose deadlines are more than  $t_0$ , but no more than  $JTP_k$ .  $L_k(t_0)$  is the number of time units in  $[0, t_0]$  that are used to run the  $PAIs$  in  $LA_k(t_0)$ .

$$PA(t_0) = \{PAI_{ij} \mid D_{ij} \leq t_0\} \quad (4)$$

$$LA(t_0) = \{PAI_{ij} \mid R_{ij} < t_0 < D_{ij}\} \quad (5)$$

$$P(t_0) = \sum_{k=1}^n [\alpha / T_k] * C_k, \text{ where } \alpha = \begin{cases} t_0 - S_k & t_0 > S_k \\ 0 & t_0 \leq S_k \end{cases} \quad (6)$$

$$LA_k(t_0) = \{PAI_{ij} \mid R_{ij} < t_0 < D_{ij} \text{ and } D_{ij} \leq JTP_k\} \quad (7)$$

**Theorem 1.**  $S(JTP_k)$  and  $S(t_0, JTP_k)$  ( $0 < t_0 < JTP_k$ ) satisfy the following equation:

$$S(t_0, JTP_k) = S(JTP_k) - t_0 + P(t_0) + L_k(t_0) \quad (8)$$

**Proof:**  $PAIs$  whose deadlines are less than  $JTP_k$  must be completed in  $[0, JTP_k]$ . Their total workload is  $P(JTP_k)$  (see Equ.4 and 6). The workload of  $P(t_0)$  and  $L_k(t_0)$  has to be 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  $NPA$ s 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 exists:  $S(t_0, JTP_k) = JTP_k - P(JTP_k) - t_0 + P(t_0) + L_k(t_0)$ . In addition,  $JTP_k - P(JTP_k) = S(JTP_k)$ , and therefore Equ.8 also holds.

Theorem 2 reveals the distribution property of the remaining time units before  $t_0$  after running  $PAIs$  in  $PA(t_0)$  as well as  $NPA$ s.  $I_p^{t_0}(t_s, t_0)$  represents the number of time units left in  $[t_s, t_0]$  after executing  $PAIs$  in  $PA(t_0)$ ;  $I_{P,A}^{t_0}(f, t_0)$  represents the number of time units left in  $[f, t_0]$  after executing both  $PAIs$  in  $PA(t_0)$  and also  $NPA$ s.

**Theorem 2.** Let the last  $NPA$  before  $t_0$  be completed at time  $f$ , then there exists such a time point  $t_s$  in  $[f, t_0]$ , that a) there are no idle slots in  $[f, t_s]$ , b) either  $PAIs$  in  $PA(t_0)$  retain the same execution pattern in  $[t_s, t_0]$  as the case when no  $NPA$ s are run before  $t_0$ , or all  $PAIs$  in  $PA(t_0)$  are completed before  $t_s$ , and c)  $t_s$  can be determined by Equ.9.

$$I_p^{t_0}(t_s, t_0) = I_{P,A}^{t_0}(f, t_0) \quad (9)$$

**Proof:** The execution of the last *NPA* may delay the execution of *PAIs* in  $PA(t_0)$ . The delayed *PAIs* may delay other *PAIs* in  $PA(t_0)$  further. The delay chain will however cease when the delayed *PAIs* no longer delay other *PAIs*, or all the *PAIs* in  $PA(t_0)$  are complete. Since all *PAIs*  $PA(t_0)$  must be complete before  $t_0$ , such a time point,  $t_s$ , must exist that satisfies Theorem 2.2. Since there are unfinished workloads before  $t_s$ , Theorem 2.1 also exists. Equ. 9 is a direct derivation from Theorem 2.1 and 2.2.

Since *PAIs* in  $PA(t_0)$  running in  $[t_s, t_0]$  retain the original execution pattern (as though there were no preceding *NPAs*), it is possible to calculate the remaining time units in  $[t_s, t_0]$  after running these *PAIs*. Consequently,  $L_k(t_0)$  in Equ.8 can be calculated.

## 4 Scheduling Algorithms

If a *NPA* starts execution at  $t_0$ , using Equ.8, the global scheduler can calculate how many idle time units there are between  $t_0$  and any *JTP* following  $t_0$ , which can be used to run the *NPA*. Therefore, it can be determined before which *JTP* the *NPA* can be completed. Consequently, the *NPA*'s finish time in any workstation  $p_j$  can be determined, which is shown in Algorithm 1. If the *NPA*'s finish time on any workstation in the heterogeneous cluster is greater than its deadline, the *NPA* will be rejected. The admission control is shown in Algorithm 2. When more than one workstation can satisfy the *NPA*'s deadline, the system selects the workstation on which the *NPA* will have the earliest finish time. After deciding which workstation the *NPA* should be scheduled to, the global scheduler re-sets the *NPA*'s deadline to its finish time on that workstation and sends the *NPA* to it. The global dynamic scheduling algorithm is shown in Algorithm 3. When the local scheduler receives the new allocated *NPAs* or the *PAIs* are ready, it inserts them into the *LSQ*. Each time a task (*NPA* or *PAI*) is fetched by the local scheduler from the head of the *LSQ* and the task is then executed. As the modeling analysis suggests in Section 3, a *NPA* cannot be finished earlier in the workstation on which the new task is scheduled. Otherwise, some *PAI*'s deadline on that workstation must be missed. In this sense, the modeling approach is optimal.

**Algorithm 1 Calculating the finish time of  $NPA_i$  starting at  $t_0$  in workstation  $p_j$**  (denoted as  $ft^j(NPA_i)$ )

1.  $c^j(cv_i) \leftarrow NPA_i$ 's execution time;
2. Calculate  $P(t_0)$ ; Get  $t_s$ ;
3. Get the first *JTP* after  $t_0$ ;
4. Call Algorithm 1 to calculate corresponding  $L_k(t_0)$ ;
5. Calculate  $S(t_0, JTP)$  using Equ.8;
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 Equ.8;
9. **end while**
10.  $ft^j(NPA_i) \leftarrow OJTP + c^j(cv_i) - S(t_0, OJTP)$ ;

**Algorithm 2 Admission Control**

1.  $PC \leftarrow \Phi$  when a new  $NPA_i$  arrives;
2. **for** each workstation  $p_j$  ( $1 \leq j \leq m$ ) **do**
3.     call Algorithm 1 to calculate  $ft^j(NPA_i)$ ;

```

4.   if ( $ft^j(NPA_i) \leq d_i$ ) then  $PC = PC \cup \{p_j\}$ ;
5. end for
6. if  $PC = \emptyset$  then reject  $NPA_i$ ;
7. else accept  $NPA_i$ ;

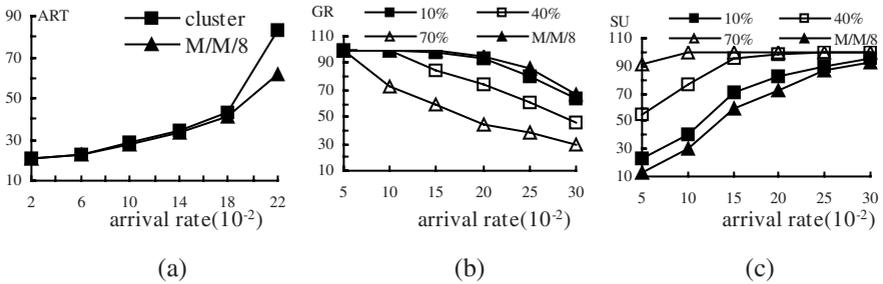
```

**Algorithm 3 the Global dynamic scheduling algorithm**

```

1. if global schedule queue  $GSQ = \emptyset$  then
   wait until a new  $NPA$  arrives;
2. else
3.   get a  $NPA$  from the head of  $GSQ$ ;
4.   call Algorithm 3 to judge if accept or reject it;
5.   if accept the  $NPA$  then
6.     select workstation  $p_j$  by response-first policy;
7.     set the  $NPA$ 's deadline to be its finish time;
8.     Dispatch the  $NPA$  to workstation  $p_j$ ;
9.   end if
10. end if
11. go to step 1;

```



**Fig. 2.** (a) Comparison of ART of  $NPA$ s between our scheduling scheme and an M/M/8 queuing model, 10% PRA, (b) Effect of workload on GR, (c) Effect of workload on SU,  $MAX\_DR/MIN\_DR=1.0/0$ ,  $MAX\_CV/MIN\_CV=25/5$ ,  $MAX\_W/MIN\_W=4/1$

## 5 Performance Evaluation

The performance of the global scheduling algorithm is also evaluated through extensive simulation. Workstation  $p_i$ 's weight  $w_i$  is chosen uniformly between  $MAX\_W$  and  $MIN\_W$ .  $NPA$ s arrive at the cluster following a Poisson process with the arrival rate of  $\lambda$ . The  $NPA_i$ 's computational volume  $cv_i$  is uniformly chosen between  $MAX\_CV$  and  $MIN\_CV$ , and the  $NPA_i$ 's deadline is chosen as follows:  $arrival-time + \min\{c^k(cv_i)\} + cv_i * nw * dr$  ( $1 \leq k \leq m$ ), where  $nw$  is the geometric mean of the weight of all workstations, and  $dr$  is chosen uniformly between  $MAX\_DR$  and  $MIN\_DR$ . Three levels of PRA workload, light, medium and heavy, are generated for each workstation, which provides 10%, 40% and 70% system utilization, respectively. Three metrics are measured in the simulation experiments: *Guarantee Ratio (GR)*, *System Utilization (SU)* and *Average Response Time (ART)*. 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. A *NPA's Response Time* is defined as the difference between its arrival time and the finish time. The *ART* is the average response time for all *NPA*s.

Fig.2.a displays the *ART* of *NPA*s as a function of  $\lambda$  in a cluster of 8 workstations each running 10% *PRA* workload. The *Guarantee Ratio (GR)* of *NPA*s is fixed to be 1.0. An *M/M/8* queuing model is used to compute the ideal bound for the *ART* of the same *NPA* workload in the absence of *PRA*s. As can be observed from Fig.2.a, the *ART* obtained by this scheduling scheme is very close to the ideal bound except that  $\lambda$  is greater than 0.18. This suggests that the scheduling scheme can make full use of the idle slots and new *NPA*s are completed at the earliest possible time.

Fig.2.b and Fig.2.c illustrate the impact of *NPA* and *PRA* workload on *GR* and *SU*, respectively. It can be observed from Fig.2.b that *GR* decreases as  $\lambda$  increases or *PRA* workload increases, as would be expected. A further observation is that the curve for 10% *PRA*, as well as the curve for 40% *PRA* when  $\lambda$  is less than 0.1, is very close to that for the *M/M/8* queuing model; which again shows the effectiveness of the scheduling scheme in utilizing idle capacities. Fig.2.c demonstrates that *SU* increases as  $\lambda$  increases. This figure shows that utilization of the cluster is significantly improved compared with the original *PRA* workload.

## 6 Conclusions

An optimal modeling approach for spare capabilities in heterogeneous clusters is presented in this paper. Furthermore, a dynamic scheduling scheme is proposed to allocate newly arriving real-time applications on the cluster by utilizing the modeling results.

## References

1. K. Hwang and Z. Xu.: Scalable Parallel Computing: Technology, Architecture, Programming. McGraw Hill, 1998.
2. 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.
3. G.R. Nudd, D.J.Kerbyson et al.: PACE-a toolset for the performance prediction of parallel and distributed systems. International Journal of High Performance Computing Applications, Special Issues on Performance Modelling, 14(3), 2000, 228–251.
4. X Qin and H Jiang.: Dynamic, Reliability-driven Scheduling of Parallel Real-time Jobs in Heterogeneous Systems. In Proceedings of the 30<sup>th</sup> International Conference on Parallel Processing (ICPP 2001), Valencia, Spain, September 3–7, 2001.
5. D.P. Spooner, SA. Jarvis, J. Cao, S. Saini and GR. Nudd.: Local Grid Scheduling Techniques using Performance Prediction. IEE Proc., Comput. Digit. Tech., 2003, 150, (2), pp. 87–96.