Research Journal of Applied Sciences, Engineering and Technology 5(24): 5614-5618, 2013
DOI:10.19026/rjaset.5.4245
ISSN: 2040-7459; e-ISSN: 2040-7467
© 2013 Maxwell Scientific Publication Corp.
Submitted: October 17, 2012 Accepted: December 19, 2012 Published: May 30, 2013

Research Article

Study on Real-time Task Scheduling Policy for Automated Testing of Spacecraft

[1]Li Zhongwen, [2]Li Long and [3]Li Guoxin
[1]College of Polytechnic, Hunan Normal University, Changsha 410081, China
[2]China Patent Technology Development Company,
[3]Computer Lab, China University of Political Science and Law, Beijing 100088, China

Abstract: Spacecraft testing is an important phase of developing a spacecraft. As the test task requests in spacecraft testing have dynamic, real-time and resource constraints characteristics, it need further research on requests scheduling policy. Based on the EDF scheduling model, this study apprehends the dynamic, real timing and communicating resource limitation in automated testing of application-level device gateway in spacecraft subsystem in its design of a multiple-factor based PRI ascertain method and put forward a scheduling policy of multithreading under resource limitation. Test results show that this policy better assures scheduling effect with effectivescheduling of each parallel testing task under different task densities.

Keywords: Automated testing, EDF, spacecraft testing, real-time task scheduling

INTRODUCTION

In automated testing of the spacecraft, the application-level device gateway connects to each downstream device-level device gateway that corresponds to spacecraft subsystem and through sending various control commands to the device-level device gateway, realized functions of power supply, management, data loading, measurement and incentive for test devices; on the other hand, provided services of sending instructions and data acquisition for the upper layer test task ends and shielded the communication protocol differences between the device-level device gateway and application-level device gateway, to provide more versatility for upper layer test task interpretation and implementation system, while the entire test system has better scalability.

Task requests that are scheduled in application-level device gateway have the following characteristics:

- **Dynamic character of task requests:** In test task ends, a number of test units perform tests in parallel and within each test task there are also parallel test operations, so for the application-level device gateway, the instruction of request sending or data acquisition issued by each test unit dynamically changes with the execution of test.
- **Real-time character of task requests:** During test, the spacecraft state is constantly changing in accordance with transmit of remote control instructions and the change of state has strict timing relationship with transmit of the remote control commands, so the test process of spacecraft is a real-time monitoring and controlling process.
- **Resource constraints:** Application-level device gateway connects to a number of spacecraft subsystems through the channel of device-level device gateway. Given parallel task requests, once a channel is occupied by a certain task request, the channel resources will not be released until the task request is completed. And this kind of communication channel resources is limited.

Therefore, we need research the scheduling policy of task requests, according to the task requests above characteristics.

TASK REQUEST SCHEDULING MODEL BASED ON EDF

The most widely used real-time task scheduling policy of uniprocessor is priority-based scheduling policy and the scheduling priority is divided into fixed-priority scheduling and dynamic priority scheduling (Liu and Layland, 1973). A typical fixed-priority scheduling includes RMS algorithm (Rate-Monotonic Scheduling) (Leung and Whitehead, 1982) and DMS algorithm (Deadlines-Monotonic Scheduling) (Narlikar *et al*., 2010). A typical dynamic priority scheduling algorithm includes EDF (Earliest-Deadline First) scheduling algorithm (Hei and Tsang, 2002) and LSF (Least Slack, First) (Jin *et al*., 2004) scheduling algorithm.

Since the task priority is fixed before scheduling in fixed priority scheduling algorithm, fixed priority

Corresponding Author: Li Zhongwen, College of Polytechnic, Hunan Normal University, Changsha 410081, China

algorithm is not applicable to task requests in application-level device gateway because of they have the character of dynamically changing. Among dynamic priority strategies, the EDF scheduling algorithm has a wide range of application (Zhu and Mueller, 2006; Choi and Kim, 2007), which assigns the priority of a certain task according to its absolute deadline. Liu and Layland (1973) have already proven that EDF scheduling algorithm is an optimal scheduling algorithm when the schedulable utilization rate is less than or equal to 1(Leung and Whitehead, 1982).

In EDF scheduling algorithm, task priority is entirely determined by the time attribute of the task, but in application-level device gateway, each test task requests come from test task end have their own value attribute, which represents the importance level of the task, therefore, in this hybrid task scheduling model, the task with smaller deadline is not necessarily scheduled and executed with higher priority and at this point, the task request's variety of attributes is needed to be considered in order to determine the task's final priority. Without loss of generality, in this study we do scheduling analysis according to the deadline attribute and value attribute of the task requests and for the other attributes, or three or more attributes can be deduced from this.

- **The formal definition of a task request:**

Firstly, the definition of task requests is given as follows:

Suppose that at time *t*, there is a set of task requests $\Gamma = \{\tau_i | 1 \leq i \leq n\}$, which contains *n* real-time task requests, with a definition of $\tau_i = \{a_i, c_i, d_i, v_i, q_i\}$, in which:

- a_i represents the arrival time of the task request, i.e., the time when the task request is activated and ready to be performed
- c_i represents the length of execution time of the task request
- d_i represents the absolute deadline of the task request, i.e., implement of the task request should be completed at this time and a valuable outcome is supposed to be produced
- r_i represents the relative deadline of the task request and $r_i = d_i - t$
- v_i represents the value of a task request, namely the criticality of task request, which means the degree of importance of the task request compared to other task requests in the set of task requests
- q_i represents the initiator of the task request

For each task request in set Γ, its final priority is determined by two parameters, namely the relative deadline r_i and value v_i of the task request.

- **Multi-factor constrained priority determining method:** Determining task priority based on multiple factors has the following two commonly used methods: linear weighted method and priority table design method.

Linear weighted method: The basic idea of linear weighted method is to use linear weighted arithmetic directly on the relative deadline and the value of the task that is, $p_i = k.r_i + (1-k).v_i$ (Jin *et al*., 2003)**Error! Reference source not found.**, in which $k \in [0,1]$ is the weighted coefficient between the two. The inadequacy of linear weighted method is: the relative deadline and the value of the task are two totally different concepts with different units of measurement, so we should not simply use weighted arithmetic on them.

- **The priority table design method:** Wang *et al*. (2004) proposed priority table design method based on PTD (Jin *et al*., 2003) for determining priority. The basic idea of priority table design method is to aggregate the two parameters of task request attributes into a two-dimensional priority table (Wang *et al*., 2004), in which the values indicate the final priority of the task request, with smaller numerical value representing higher priority of the task request. Priority table design method eliminates the problem of unable to get direct dealing and calculation caused by the dimensionless of task requests, but it still has drawbacks as follows:
- The design of priority table is relatively cumbersome and along with every scheduling analysis, the priority table should be redesigned, so when handling more tasks requests the calculation amount of scheduling analysis will increase
- The design rules of priority table are more fixed, when it is needed to increase the weight of a certain parameter in scheduling process, it can not be effectively expressed by the method.

This study integrated the designing ideas of the two methods above, proposed linear weighted priority table method to determine priority for the task requests, which is described as follows:

Sort all the task requests in task request set in accordance with their relative deadline and value, to obtain relative deadline sequence *seriR* = $(r_1, r_2, \ldots, r_n)$, where $r_1 < r_2 <, \ldots, < r_n$ and task request value sequence *seriV* = $(v_1, v_2, \ldots, v_n)$, where $v_1 > v_2 > \ldots > v_n$. The two attributes r_i and v_i of each task request τ_i in the set of task request Γ are inevitably corresponding to two values in the above-mentioned two sequence of numbers, denoted as $<r_i, v_i> \rightarrow <m, n>$, where *m*, *n* presents the serial number of the position that r_i and v_i corresponds to respectively in the sequence *seriR* and sequence *seriV* and then based on the idea of linear weighting, obtained the expression of task request's final priority:

$$p = k \cdot m + (1-k) \cdot n \tag{1}$$

The weighted priority table method is characterized by:

- Eliminate the effect of task request parameters' dimensionless on priority design and made the meaning of priority expression more clear
- Compared with the priority table design method, it is not needed to establish priority table to get the final priority by successive queries and thus easier
- Priority design is more flexible by choosing different weights k to adjusted the effect of a task request attribute on priority

For three or more attributes, the basic idea of priority design are the same, except that it is needed to add one dimension or a number of dimensions and sort the variety of attributes, while adding the corresponding weighting parameters.

RESOURCE CONSTRAINED REAL-TIME TASK REQUEST SCHEDULING POLICY

After determined the method for determining task request's priority, the real-time task request scheduling algorithm in application-level device gateway is provided below: resource constrained dynamical priority scheduling algorithm based on multithreading.

The basic idea of the algorithm is: to add all of the task requests that have arrived at the present time to the implement task request list sequentially according to their priority order, at the meantime subtract the resources consumed by each task request from the total system resource set and the task requests whose resource request can not be met, are put into wait queue to wait for the next scheduling. During the above described process, the task requests that exceed their own deadline need to be constantly deleted and all of the resources occupied by task requests that have been scheduled and completed also need to be released.

Algorithm description: For task request set $\{req_i|1\leq i\leq N\}$ within time $[t, t+\Delta t]$, the task request linked list is generated based on the arrival time of each task request, while the task request execution linked list and the task request deletion linked list are initialized.
The data structure description:

- *ListExe* represents linked list of task requests being scheduled and executed
- *List Del* represents linked list of task requests to be deleted
- *List Wait* represents linked list of task requests whose resource request can not be met and are in a wait state
- *List Ready* represents linked list of task requests that the system received within the time Δt

Algorithm: *Priority Schedule*
Input: linked list of task requests being scheduled, denoted by *list*, which contains *k* task requests.
Start

Step1: At current time t_{cur}, for each task request $\tau_i = \{a_i, c_i, d_i, v_i, q_i, RS\}$ in task request linked list *list,* if d_i- $c_i \geq t_{cur}$, then remove task request τ_i from *list* and also remove all the task requests with a initiator attribute q_i, meanwhile, add all the removed task requests into linked list *listDel*
Step 2: According to formula (1), calculate priority value of each task request in *list*;
Step 3: Sort the task requests in *list* from small to large in accordance with their priority value (if a few task requests have the same priority value, then sort these task requests according to the arrival order based on a_i)
Step 4: Acquire task request $\tau_i = \{a_i, c_i, d_i, v_i, RS\}$ from the list head of *list* and do resource allocation operation on τ_i, if the allocation is successful, go to Step 5; otherwise, go to Step 6
Step 5: Open thread and execute task request τ_i, remove task request τ_i from *list*, meanwhile, add τ_i into linked list *listExe*
Step 6: Add task request τ_i into linked list *listWait* and remove it from list linked list *list*
Step 7: Determine whether the linked list *list* is empty. If it is empty, the algorithm terminates; otherwise go to Step4

End
The main algorithm is as follows:
Algorithm: main flow of scheduling

Input: None
Start

Step 1: Denote the time when scheduling manager started task request scheduling as t_{st}
Step 2: Add the received task requests into linked list list Ready
Step 3: Denote the current time of system as t_{cur}

- **In the case of $t_{cur}<t_{st}+\Delta t$:** If *listExe* is empty or there is not any execution of task request in execution linked list *listExe* is completed, go to Step 2. If there are task requests in linked list *listExe* are executed and completed, go to Step 4.
- **In the case of $t_{cur}\geq t_{st}+\Delta t$:** If *listExe* is empty or there is not any execution of task request in execution linked list *listExe* is completed, go to Step 5; If there are task requests in linked list *listExe* are executed and completed, go to Step 4.

Step 4: Construct linked list *list* and store the task requests in linked list *list ready* and linked list *list Wait* into *list*; then empty the linked list *list Ready* and linked list *list Wait*; assign t_{st} as the current system time. Execute *Priority Schedule* algorithm and use *list* as the input of algorithm *Priority Schedule*; go to Step 2

Step 5: Construct linked list *list* and store the task requests in linked list *listReady* into *list*; then empty the linked list*list Ready*; assign t_{st} as the current system time. Execute *Priority Schedule* algorithm and use *list* as the input of algorithm *Priority Schedule*; go to Step 2

End

EXPERIMENTS AND ANALYSIS

Experimental environment includes 5 device servers, 1 scheduling server and 1 execution server, each server is located in the same LAN (Table 1).

Compare the simulation results of dynamic priority scheduling policy proposed in this study with the EDF priority scheduling policy.

Experimental methods:

Execution server constantly sends out task requests $\tau_i = \{a_i, c_i, d_i, v_i, q_i, RS_i\}$ and the varieties of parameters are generated according to the following rules:

- The task execution time c_i is randomly selected between 100 milliseconds to 20,000 milliseconds, which obeys uniform distribution;
- The absolute deadline of a task: the length of delay time is $b_i = 1.3 \times c_i$and the absolute deadline of a task is determined by the time of generating the task request plus the delay time b_i;
- The task value v_i is randomly select between 1-100, which obeys uniform distribution and all task requests are divided into different test tasks q_i ($0 \le q_i \le 19$), where greater k implies higher level of criticality the test task has and a task request belongs to the k-th test task if and only if $q_i \times 10 \le q_i \le (q_i + 1) \times 10$
- The number of channel resource between each device server and the scheduling server is 10. Each task randomly selects two device servers and then randomly generates 0 or 1 channel resource request for the two device server
- The running time is 10 min and with a cycle of 5 sec, the execution server has 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100 task requests respectively generated and sent to the scheduling server. The task generation density is$\rho = \frac{tl}{pr}$, where *tl* is the number of task requests generated, *pr* is the cycle of generating task.
- Investigate the following indicator:

Table 1: Experimental environment

Experimental equipments	Hardware configuration	Software configuration
Device server	CPU: Intel Core Duo 2.6GHz Memory: 2GB	Windows XP JDK1.6
Scheduling server	HP E4440 Server Memory: 8G	HP UNIX 11.0 JDK1.6
Execution server	CPU: Intel core duo 2.6GHz Memory: 2GB	Windows XP JDK1.6

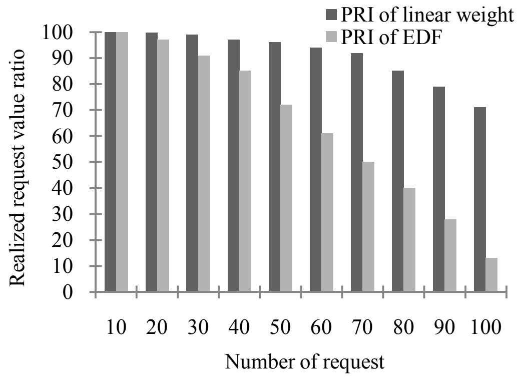

Fig. 1: Realized request value ratio at different request numbers

Realized request value ratio:

$$AVR = 100 \times \frac{\sum_{j=1}^{j=100} (j \times T_S^j)}{\sum_{j=1}^{j=100} (j \times T_T^j)} \tag{2}$$

In formula (2),T_S^j is the number of task requests with value *j* that are successfully scheduled, T_T^j is the total number of task requests with value *j*. The request value ratio reflects the ratio of task request value that can be realized by the scheduling policy in the total value of all task requests. It should be noted that when investigating the realized value ratio, only task request τ_i is deleted from the *list* in Step 1 of the scheduling policy *Priority Schedule* algorithm, with other initiators' task request with *qi* attribute remain undeleted, for we do not consider this *qi* attribute.

Form Fig. 1 we can see that when the method brought up by this study is adopted at a task density of 10, i.e., 50 tasks generated every 5 sec, the ARV ratio is over 90% and along with the increase of task density, the ARV ratio goes down. However, its overall performance is always higher than the adoption of PRI of EDF method.

Realized test task value ratio:

$$TVR = 100 \times \frac{\sum_{j=1}^{j=100} (j \times T_S^j)}{\sum_{j=1}^{j=100} (j \times T_T^j)} \tag{3}$$

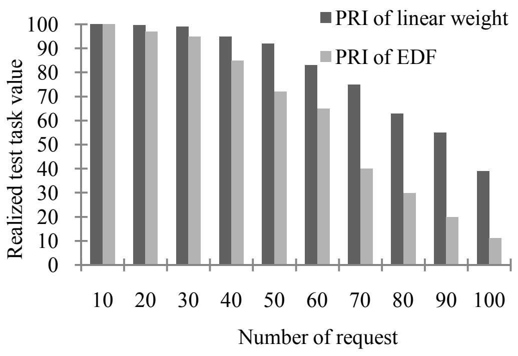

Fig. 2: Realized test task value ratio at different request numbers

In formula (3), T_S^j is the number of task requests with value *j* that are successfully scheduled, T_T^j is the total number of task requests with value *j* and the realization ratio of tested tasks is the ratio of realized tested task value in the total tested task value, which reflects that the scheduling policy can properly handle every important test task. Different form the above mentioned realized request value ratio, here the task request attribute q_i is considered, so τ_i is deleted from the *list* at Step 1 of *Priority Schedule* algorithm, together with any other initiators task request with q_i attribute.

From Fig. 2 we can see that, as EDF considers only the closed line of requests, while the method brought up by this study takes not only the closed line of requests, but also value of the request, resource required and time length of task execution into consideration, which makes it excel the EDF method in performance at different task density.

CONCLUSION

Based on the EDF scheduling model, this study apprehends the dynamic, real timing and communicating resource limitation in automated testing of application-level device gateway in spacecraft subsystem in its design of a multiple-factor based PRI ascertain method and put forward a scheduling policy of multithreading under resource limitation. Test results show that this policy better assures scheduling effect with effective scheduling of each parallel testing task under different task densities.

REFERENCES

Choi, Y. and H. Kim, 2007. A new scheduling scheme for high-speed packet networks: Earliest-virtual-deadline-first [J]. Comput. Commun., 30(10): 2291-2300.

Hei, X. and D. Tsang, 2002. Earliest deadline first scheduling with active buffer management for real-time traffic in the Internet [J]. Telecommun. Syst., 19(3-4): 349-359.

Jin, H., H.A. Wang, Q. Wang and G.Z. Dai, 2003. An integrated design method of task priority. J. Softw., 14(3): 376-382.

Jin, H., H.A. Wang, Q. Wang and G.Z. Dai, 2004. An improved least-slack-first scheduling algorithm. J. Softw., 15(8): 1116-1123.

Leung, J. and J. Whitehead, 1982. On the complexity of fixed-priority scheduling of periodic, real-time tasks. Perform. Eval., 2(12): 237-250.

Liu, C. and J. Layland, 1973. Scheduling algorithms for multiprogramming in real-time systems. J. ACM, 20(l): 46-61.

Narlikar, G., G. Wilfong and L. Zhang, 2010. Designing multihop wireless backhaul networks with delay guarantees [J]. Wirel. Netw., 16(l): 237-254.

Wang, Y.Y., Q. Wang, H.A. Wang, H. Jin and G.Z. Dai, 2004. A real-time scheduling algorithm based on priority table and its implementation. J. Softw., 15(3): 360-370.

Zhu, Y. and F. Mueller, 2006. Exploiting synchronous and asynchronous DVS for feedback EDF scheduling on an embedded platform [J]. ACM T. Embed. Comput. Syst., 5: l-24.