Petri Net Based Software Testing Scheduling and Selecting

Senke Ding
Institute of Cyber-systems and Control
Zhejiang University
Hangzhou, China
Email: dingsenke@zju.edu.cn

Peifu Xu
China Tobacco
Zhejiang Industrial Co., Ltd
Hangzhou, China
Email: xpf_nb@163.com

Weimin Wu, Yi Yang, Zichao Xing, Feihua Lu, Cheng Li
Institute of Cyber-systems and Control
Zhejiang University
Hangzhou, China
Email: wmwu@iipc.zju.edu.cn

***Abstract*—Computer software system has a profound impact on human society. It increasingly highlights the importance of software testing. Reducing the cost and improving the efficiency of software testing has an important practical significance and economic value. This paper investigates on software testing workflow from the perspective of discrete event dynamic systems and presents a method to improve the efficiency of software testing by optimizing task scheduling and execution priorities. We developed a simulation program of task scheduling based on Petri net to compare the performance of each scheduling option in different situations and made the analysis of their differences.**

I. Introduction

Information technology is playing an increasingly important role in China's economic transformation. Software system is one of the key technologies and has an influence on every aspect of people's daily life. However, in several situations software would run into issues and result in something that is different from what people expect. In severe cases, it may lead to system crashes, loss of users' funds, security risks and other problems. In order to ensure the quality, testing is a necessary process in the workflow of software development. The practice in software industry shows that, in average, software testing or system testing takes more than half of the time of the project and more than 50% of the total cost [1]. With the growing diversity of software use scenarios, degrees of software complexity and technologies of software development, traditional software testing techniques and test tools would not be able to meet needs in reality [2]. Researching on software testing is becoming more significant and difficult for academia and industry.

This paper researches on the manual testing process that is the bottleneck of the overall performance of software testing workflow. It uses Petri Net [3] as a tool, analyzes discrete event dynamical systems theory in-depth, and targets improving resource utilization without increasing labor costs and enhancing the efficiency of software testing.

The rest of this paper is organized as follows. Section 2 proposes Petri Net based software testing scheduling and selecting options. Section 3 provides some examples and the result of simulation. Finally, conclusions are given in Section 4.

This work was supported in part by the National Nature Science Foundation of China under Grant No.61374066 and 91546103, Science Fund for Creative Research Groups of China under Grant No.61621002, the Research Project of the State Key Laboratory of Industrial Control Technology, Zhejiang University, China (No.ICT1607), and the National Key Research and Development Program of China under No. 2016YFC0800105.

II. Shared Resource Software Testing Scheduling and Selection

In modern software companies, it is usual that multiple product lines are developed at the same time and that test engineers are divided into several groups to test different products. This paper mainly focuses on how to assign test tasks wisely and how to choose test tasks for each test group appropriately to achieve the most efficient overall performance, in the situation of multiple testing tasks and multiple test groups. The problem of assigning tasks is called Test Task Scheduling, and the problem of prioritizing tasks for each test group is called Test Task Selecting. The situation that each test group only executes certain types of tasks and does not execute other tasks even if idle is called Non-shared Resource Software Testing, which does not have Test Task Selecting problem. The situation that each test group can execute various types of tasks is called Shared Resource Software Testing, in which each test group can be assigned certain type of tasks according to the strategy and priority.

In the large-scale software development and testing process, a program is often divided into multiple modules. In this paper, the testing for a module is called a task; multiple pending tasks builds a queue; one person or a group of people who execute the module testing is called a test resource. This section discusses how to schedule tasks when there are multiple queues and multiple resources. Suppose the model of multi-queues and multi-resources satisfies the following conditions($1 \leq i \leq n$) [4]:

(1) The test system includes multiple queues. Each test queue is denoted by q_i.

978-1- 5090-4429- 0/17/$31.00 ©2017 IEEE

(2) A test resource cannot unlimitedly accept tasks, so each queue has a limited capacity. The capacity of a queue q_i is denoted by b_i.

(3) Test tasks are divided into n task groups. The tasks in Group i arrive following Poisson process [5] with the arrival rate λ_i. When the queue that receives the tasks from this task group is full, it stops accepting.

(4) There are n test resources, each one denoted by s_i. Its testing rate - the number of tasks completed per unit time is μ_i. Suppose testing rates are independent from each other and are exponential distribution [6].

This paper designs the model of multi-queues and multi-resources using Generalized Stochastic Petri nets (GSPN) [7]. In GSPN model, the arrival and the execution of tasks can be represented by timed transitions and the rate of execution is relevant to the system status. A task being assigned to a queue can be represented by an immediate transition. An immediate transition does not take time, but a random switch needs to be set to determine the probability of excitation. A default random switch indicates the probability of excitation is 1. A queue is represented by place and the length of a queue is represented by the mark of the place. The firing condition of transition is specified by the predicate. If the predicate is not satisfied, transition cannot be fired. A default predicate is always satisfied. A task or a resource is represented as a token.

The GSPN model of a simple task scheduling is shown in Fig. 1. In this paper, the assignment of tasks is represented by the conflicts in Petri nets; the mathematical expression of scheduling and the probability distribution are represented by the predicate of immediate transition in GSPN and random switch; the execution of task is represented by timed transition in GSPN; and various scheduling options are represented by the priority of transition's firing [8]. To simplify the scheduling option, Fig. 1 contains only two queues.

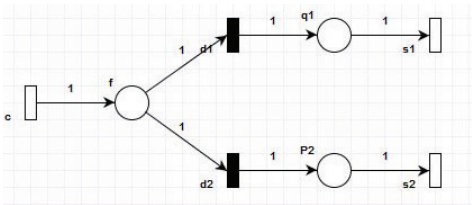

Fig. 1. Non-shared Resource Software Testing scheduling GSPN model

Fig. 2 is a basic shared resource SPN model. Place r represents the shared resource. It does not allow transitions d_1 and d_2 to be fired simultaneously. Therefore, it needs to choose one of the transitions to fire and this process is called a selection option.

Combining Fig. 2 and Fig. 1, a shared resource software testing task scheduling model is shown in Fig. 3. To be consistent, assume that the model contains n types of tasks and m testing resources. The transitions and places are ($1 \leq i \leq n$, $1 \leq j \leq m$):

c_i: Indicating the timed transitions of task arrivals. Its implementation rate is λ_i; the enable predicate is $\sum_{k=1}^{m} M(q_{ik}) < \sum_{k=1}^{m} b_{ik}$. When the queue that receives the tasks is full, it stops accepting more tasks.

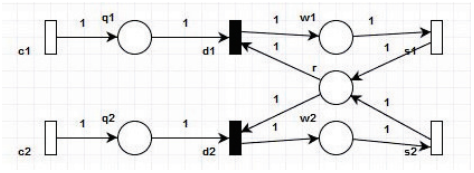

Fig. 2. Shared Resource GSPN model

f_i: Used to determine task assignments. When i-th types of tasks arrive, the queue that the tasks get into is determined by the enable predicate associated with d_{ij} or the random switch.

d_{ij}: Representing the execution of scheduling, which is an immediate transition indicating the task assignment doesn't take any time. The enable predicates and the random switch of d_{ij} represent the scheduling options.

q_{ij}: A first-in first-out queue for pending tasks. Its capacity is limited to b_{ij}

s_{ij}: Representing the testing resource whose testing rate is μ_{ij}. Note that all s_{kj} ($1 \leq k \leq n$) represents one resource executing different tasks with sharing the resource j. In order to make the model concise, the structure of shared testing resources is not shown in Fig. 3.

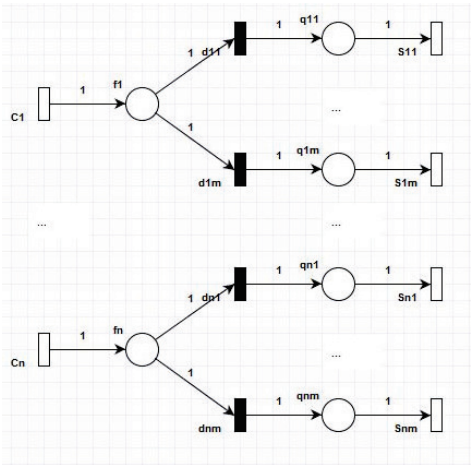

Fig. 3. Shared Resource Software Testing scheduling and selecting GSPN model

The enable predicates and random switch associated with transitions d_{ij} represent task scheduling options. The enable predicates and random switch associated with s_{ij} represent shared resource selecting options. There are three selecting options in the following:

1) Random Selecting (RS)
The enable predicate associated with transition s_{ij} is default. Thus the predicate condition is always true.
The testing rate of s_{ij} is x_{ij}

$$x_{ij}(M) = \begin{cases} \frac{1}{|RS(M)|} \times \mu_{ij}, & if \quad i \in RS(M) \\ 0, & else \end{cases} \quad (1)$$

where $RS(M) = \{k \,|M(q_{kj}) > 0, 1 \leq k \leq n\}$
This option means that the resource s_{ij} randomly selects tasks to execute from a non-empty queue.

2) Longest Queue Selecting (LQS)
The enable predicate associated with transition s_{ij} is default. The testing rate of s_{ij} is x_{ij}

$$x_{ij}(M) = \begin{cases} \frac{1}{|LQS(M)|} \times \mu_{ij}, & if \quad i \in LQS(M) \\ 0, & else \end{cases} \tag{2}$$

where $LQS(M) =$
$\{k \,|M(q_{kj}) = \max[M(q_{1j}), M(q_{2j}), \ldots, M(q_{nj})]\}$
This option means that the resource s_{ij} selects the longest queue to make it as short as possible. If there exist more than one longest queues, then the probability of each queue being executed is even.

3) Shortest Service Time Selecting (SSTS)
The enable predicate associated with transition s_{ij} is default. The testing rate of s_{ij} is x_{ij}

$$x_{ij}(M) = \begin{cases} \frac{1}{|SSTS(M)|} \times \mu_{ij}, & if \quad i \in SSTS(M) \\ 0, & else \end{cases} \tag{3}$$

where

$$SSTS(M) = \{k | \mu_{kj} = \max[\mu_{1j}, \mu_{2j}, \cdots, \mu_{nj}]\}$$

Since one resource has various rates to execute different tasks, $\mu_{ij} \neq \mu_{kj} (i \neq j)$. The option means that the resource s_{ij} can select a queue where it has the maximum testing rate, so that it can reduce the time spent on executing as much as possible. If there exist more than one such queues, then the probability of each queue being selected is even.

There are four scheduling options:

1) Random Routing (RR)
The enable predicate associated with transition d_{ij} is y_{ij}

$$y_{ij} : M(q_{ij}) < b_{ij} \tag{4}$$

The random switch g_{ij} associated with transition d_{ij}

$$g_{ij}(M) = \begin{cases} \frac{1}{|RR(M)|}, & if \quad j \in RR(M) \\ 0, & else \end{cases} \tag{5}$$

where$RR(M) = \{k \,|M(q_{ik}) = b_{ik}\}$

2) Shortest Queue Routing (SQR) The enable predicate associated with transition d_{ij} is y_{ij}

$$\begin{aligned} y_{ij} :& (M(q_{ij}) < b_{ij}) \wedge (for \quad \forall k \neq j, \\ & (M(q_{ij}) \leq M(q_{ik})) \vee (M(q_{ik}) = b_{ik})) \end{aligned} \tag{6}$$

The random switch g_{ij} associated with transition d_{ij}

$$g_{ij}(M) = \begin{cases} \frac{1}{|SQR(M)|}, & if \quad j \in SQR(M) \\ 0, & else \end{cases} \tag{7}$$

where

$$SQR(M) = \left\{ k \,\middle|\, \begin{array}{l} M(q_{ik}) = \min(M(q_{i1}), M(q_{i2}), \\ \cdots, M(q_{im})) and (M(q_{ik}) < b_{ik}) \end{array} \right\}$$

3) Shortest Expected Delay Routing (SEDR) The enable predicate associated with transition d_{ij} is y_{ij}

$$\begin{aligned} y_{ij} :& (M(q_{ij}) < b_{ij}) \wedge (for \quad \forall k \neq j, \\ & (M(q_{ij})/\mu_{ij} \leq M(q_{ik})/\mu_{ik}) \\ & \vee (M(q_{ik}) = b_{ik})) \end{aligned} \tag{8}$$

The random switch g_{ij} associated with transition d_{ij}

$$g_{ij}(M) = \begin{cases} \frac{1}{|SEDR(M)|}, if \quad j \in SEDR(M) \\ 0, \quad else \end{cases} \tag{9}$$

where

$$SEDR(M) = \left\{ k \,\middle|\, \begin{array}{l} \frac{M(q_{ik})}{\mu_{ik}} = \min(\frac{M(q_{i1})}{\mu_{i1}}, \frac{M(q_{i2})}{\mu_{i2}}, \\ \cdots, \frac{M(q_{im})}{\mu_{im}}) and (M(q_{ik}) < b_{ik}) \end{array} \right\}$$

4) Overall Shortest Expected Delay Routing (OSEDR) The enable predicate associated with transition d_{ij} is y_{ij}

$$\begin{aligned} y_{ij} :& (M(q_{ij}) < b_{ij}) \wedge \\ & ((\sum_{y=1}^{n} \frac{M(q_{yk})}{\mu_{yk}} = \\ & \min(\sum_{y=1}^{n} \frac{M(q_{y1})}{\mu_{y1}}, \sum_{y=1}^{n} \frac{M(q_{y2})}{\mu_{y2}}, \cdots, \sum_{y=1}^{n} \frac{M(q_{ym})}{\mu_{ym}})) \vee \\ & (for \forall k \neq j, M(q_{ik}) = b_{ik})) \end{aligned} \tag{10}$$

The random switch g_{ij} associated with transition d_{ij}

$$g_{ij}(M) = \begin{cases} \frac{1}{|OSEDR(M)|}, & if \quad j \in OSEDR(M) \\ 0, & else \end{cases} \tag{11}$$

where

$$OSEDR(M) = \left\{ k \,\middle|\, \begin{array}{l} \sum_{y=1}^{n} \frac{M(q_{yk})}{\mu_{yk}} = \\ \min \left[\sum_{y=1}^{n} \frac{M(q_{y1})}{\mu_{y1}}, \sum_{y=1}^{n} \frac{M(q_{y2})}{\mu_{y2}}, \cdots, \sum_{y=1}^{n} \frac{M(q_{ym})}{\mu_{ym}} \right] \\ and (M(q_{ik}) < b_{ik}) \end{array} \right\}$$

III. Simulation and Result Analysis

In this paper, a program based on Petri nets is developed in Python to simulate task scheduling and selecting. The program has three classes that represent Petri nets' elements: *Place*, *ImmediateTransition*, *StochasticTransition*, which represent place, immediate transition and timed transition respectively. The structure is shown in Fig. 4. The class member variables *parents* and *children* are used to save network structure. *Parents* represent the set before node and *children* represent the set after nodes. There is also a class called *Simulation*. It has a member function *__init__(unit)* is used to creating the scheduling model, in which *unit* is the running cycle of the simulation program. Another member function called *Start(routing, selecting)* would start to run the program. The parameter *routing* represents the scheduling option, which is one of four options that were mentioned above, RR, SQR, SEDR and OSEDR; the other parameter *selecting* represents

the selecting option, which is one of the three options that were also mentioned above, RS, LQS and SSTS.

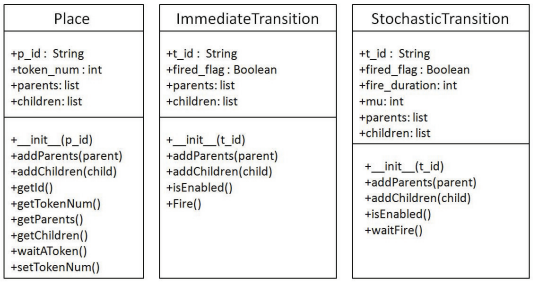

Fig. 4. Class diagram of Petri Nets based task scheduling and selecting simulation program

Suppose a company has two product lines, called Product 1 and Product 2, and two groups of test engineers. Due to the differences of product development progress, tester engineers are not only working on one of the product lines, but could possibly work on either or both of them. The model in Fig. 5 was created based on it and the meaning of each place and transition was shown in Table I and Table II.

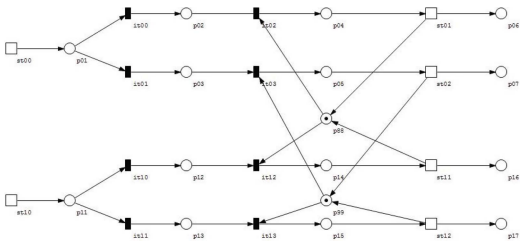

Fig. 5. Instance of Petri nets based scheduling and selecting model

TABLE I
THE MEANING OF PLACES IN PETRI NETS BASED SCHEDULING AND SELECTING MODEL

Place	Meaning
p_{01}	Test task of Product 1 is pending for assignment
p_{02}	Test task of Product 1 is in the queue of Test Group 1
p_{03}	Test task of Product 1 is in the queue of Test Group 2
p_{04}	Test task of Product 1 is being executed by Test Group 1
p_{05}	Test task of Product 1 is being executed by Test Group 2
p_{06}	Test task of Product 1 was completed by Test Group 1
p_{07}	Test task of Product 1 was completed by Test Group 2
p_{11}	Test task of Product 2 is pending for assignment
p_{12}	Test task of Product 2 is in the queue of Test Group 1
p_{13}	Test task of Product 2 is in the queue of Test Group 2
p_{14}	Test task of Product 2 is being executed by Test Group 1
p_{15}	Test task of Product 2 is being executed by Test Group 2
p_{16}	Test task of Product 2 was completed by Test Group 1
p_{17}	Test task of Product 2 was completed by Test Group 2

In the model, a token represents a test task; the timed transition st_{00} and st_{10} are in the enable state all the time, which means the task continues to arrive with the rates λ_0 and λ_1; the arrival of tasks is Poisson distribution. Timed transition $st_{01}st_{02}st_{11}st_{12}$ has the test rates $\mu_{01}\mu_{02}\mu_{11}\mu_{12}$ respectively and they are all exponentially distributed. In this paper, the scheduling and selecting options are evaluated by the task completion rate. Task completion rate ρ is equal to the ratio of the number of completed tasks to the total number of tasks, which is represented by the formula below (ignore the tasks that are being executed considering they are much fewer than the ones that have been completed or the ones that are pending), where $TN(p_{ij})$ is the number of tokens at place p_{ij}.

$$\rho = [TN(p_{06}) + TN(p_{07}) + TN(p_{16}) + TN(p_{17})] / \left[\begin{array}{l} TN(p_{06}) + TN(p_{07}) + TN(p_{16}) + TN(p_{17}) \\ +TN(p_{02}) + TN(p_{03}) + TN(p_{12}) + TN(p_{13}) \end{array} \right]$$

TABLE II
THE MEANING OF TRANSITIONS IN PETRI NETS BASED SCHEDULING AND SELECTING MODEL

Transition	Meaning
st_{00}	Test task of Product 1 arrives
st_{01}	Test Group 1 is testing test task of Product 1
st_{02}	Test Group 2 is testing test task of Product 1
it_{00}	Test task of Product 1 is assigned to Test Group 1
it_{01}	Test task of Product 1 is assigned to Test Group 2
it_{02}	Test Group 1 selects test task of Product 1
it_{03}	Test Group 2 selects test task of Product 1
st_{10}	Test task of Product 2 arrives
st_{11}	Test Group 1 is testing test task of Product 2
st_{12}	Test Group 2 is testing test task of Product 2
it_{10}	Test task of Product 2 is assigned to Test Group 1
it_{11}	Test task of Product 2 is assigned to Test Group 2
it_{12}	Test Group 1 selects test task of Product 2
it_{13}	Test Group 2 selects test task of Product 2

Next, a variety of circumstances that could affect the task completion rate of scheduling options and selecting options are analyzed. The period of the simulation cycle is set to 10,000 units, which is sufficient to make the system to reach a stable state. Since the task arrival rates and test rates are randomly distributed, in order to make the results more convincing, the following task completion rates are the average value of 10 times' simulation results.

1) $\lambda_0 = \lambda_1$, $\mu_{01} = \mu_{02} = \mu_{11} = \mu_{12}$.The arrival rates of two types of tasks are equal and the test rates of two test groups are equal.
   The simulation results are shown in Fig. 6. X-axis is defined in the figure, which is the ratio of task arrival rate to the overall test rate. It is shown in the figure that when the task arrival rate is much less than the test rate, the task completion rate is nearly 100%. With the arrival rate gradually approaching the overall test rate, the task completion rate is gradually decreasing. It meets the situation in reality that when the task arrival rate becomes larger and larger, test groups have been working at full capacity. So the pending tasks start to pile up. In the figure, 12 curves' trends are consistent, which indicates that when two types of tasks arrive at the same rate and each test group has the same test rate for two types of tasks, there is little difference between different scheduling and selecting options.

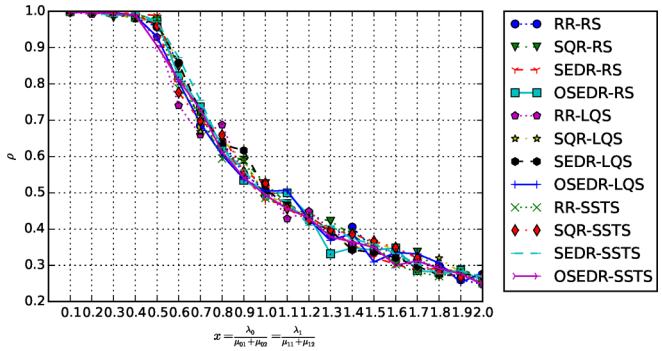

Fig. 6. Simulation Result 1

2) $\lambda_0 = \lambda_1$, $\mu_{01} = 5 \times \mu_{02}$, $5 \times \mu_{11} = \mu_{12}$. The arrival rates of two types of tasks are equal. Test Group 1 is good at testing Product 1 and Test Group 2 is good at testing Product 2. There is an obvious difference between the test rates.
The simulation results are shown in Fig. 7. It shows that SEDR- LQS, SQR-SSTS and SEDR-SSTS options have the best performance. The task completion rate decreases slowly and remains above 90% all the time. RR-SSTS and OSEDR-SSTS options perform a bit worse and the others have poor performance. Since the arrival rates of two types of tasks are equal, different scheduling options have little impact on the task completion rate, while selecting options have a greater impact. Overall, the selecting option SSTS has the advantage compared to RS and LQS because SSTS allows the test groups to first execute the tasks they are good at.

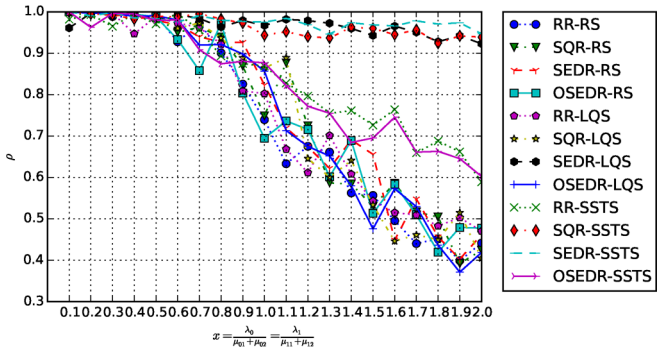

Fig. 7. Simulation Result 2

3) $\lambda_0 = 3 \times \lambda_1$, $\mu_{01} = 3 \times \mu_{02}$, $3 \times \mu_{11} = \mu_{12}$. The arrival rates of two types of tasks have an obvious difference. Test Group 1 is good at testing Product 1 and Test Group 2 is good at testing Product 2. There is an obvious difference between the test rates.
The simulation results are shown in Fig. 8 . As the arrival rates and the test rate are quite different, both scheduling and selecting options have a great impact on the task completion rate. Firstly, for the impact of the scheduling options on the task completion rate, it's obvious that RR has the worst performance. Because this option does not take into account the state of the system. There are a large number of tasks to get into the queue that is executed at a slow rate. It results in the slow queue getting longer and longer and the fast queue running out of tasks. Among the curves of RR scheduling options, the curve of selecting option LQS has the worst performance. This is because the queue being executed slowly is always longer than the queue being executed fast and LQS option always selects the task from the longer queue. Then the test groups are always assigned to the tasks that they are not good at, which results in the lowest task completion rate. SEDR is the best option, because it takes into account the test rates of the different tasks. Then each task is assigned to the group who are good at it. The best combinations of scheduling and selecting options are SEDR-LQS, SEDR-SSTS and SQR-SSTS, as either each task is assigned to a more appropriate test group, or the test group selects a more appropriate task. That's why they achieve the higher task completion rates.

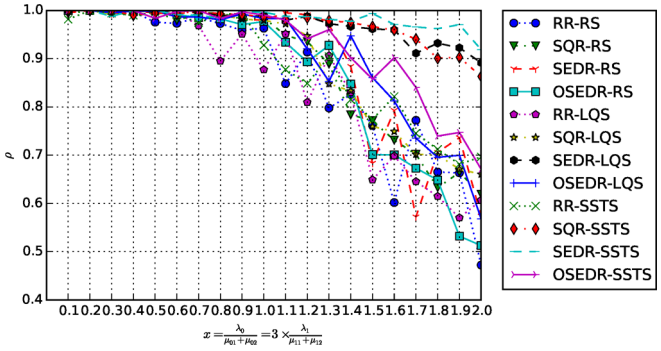

Fig. 8. Simulation Result 3

4) $\lambda_0 = \lambda_1$, $\mu_{01} = 5 \times \mu_{02}$, $\mu_{11} = 5 \times \mu_{12}$. The arrival rates of two types of tasks are equal. Test Group 1 consists of senior test engineers who have a larger test rate of executing both types of tasks, while Test Group 2 consists of junior test engineers who have a smaller test rate of executing both types of tasks.
The simulation results are shown in Fig. 9. Test Group 1 has the same test rate for two types of tasks as well as Test Group 2, while Test Group 1 performs faster than Test Group 2. So selecting options have less effect than scheduling options on the task completion rate. Since RR option randomly assigns task, a number of tasks are assigned to Test Group 2, while other scheduling options tend to assign more tasks to Test Group 1. Thus RR is less effective than others.
5) $\lambda_0 = \lambda_1$, $\mu_{01} = \mu_{02} = 5 \times \mu_{11} = 5 \times \mu_{12}$. The arrival rates of two types of tasks are equal, but they have various difficulties. Testing Product 1 is less difficult as it is more stable. Testing Product 2 is more difficult as it

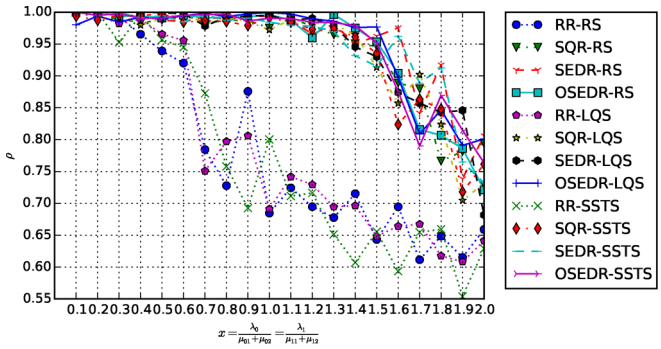

Fig. 9. Simulation Result 4

is newly developed and has more bugs. So both groups have a faster test rate for Product 1 than Product 2.

The simulation results are shown in Fig. 10. It shows that scheduling options have little effect on the task completion rate and selecting option SSTS has a significantly better performance than others. With SSTS option both test groups tend to choose the tasks for Product 1 which can be executed faster. Therefore, more tasks can be completed.

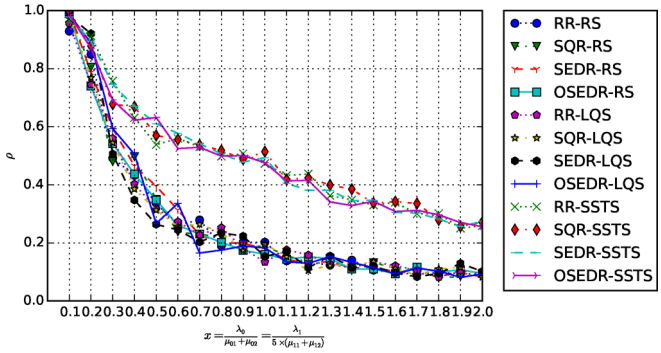

Fig. 10. Simulation Result 5

6) $\lambda_0 = 3 \times \lambda_1$, $\mu_{01} = \mu_{02} = \mu_{11} = \mu_{12}$. Product 1 has a faster arrival rate and both test groups have the same test rate of two types of tasks.

The simulation results are shown in Fig. 11 . Each scheduling option and selecting option does not have a big difference.

According to the simulation results, there are the following conclusions:

(1) In the condition that each test group has the same test rate for executing both types of tasks, scheduling options and selecting options have little impact on the task completion rate.

(2) The task completion rate depends on scheduling options in some conditions and depends on selecting options in some conditions. However, it's more likely that a combination of both options is needed to achieve a good performance.

(3) Overall, SEDR scheduling option and SSTS selecting option have a better robustness, which are more efficient in various situations.

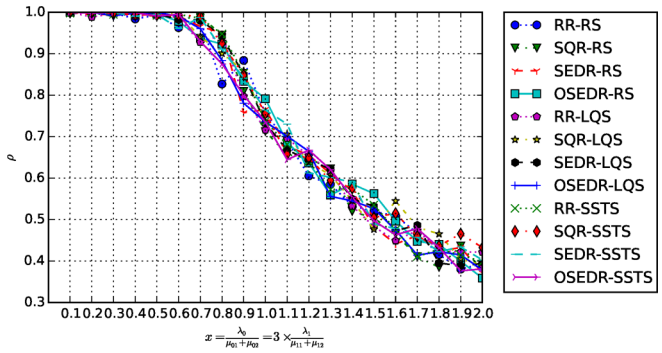

Fig. 11. Simulation Result 6

(4) Evaluating scheduling and selecting options with task completion rate has limitations, such as in simulation results 5, the tendency to testing Product 1 rarely occur in practice. One solution is to add the task weight factor in scheduling and selecting options, so that the result can be adjusted according to the urgency of the task in reality.

IV. Conclusion

In this paper, the software testing workflow model is presented using Generalized Stochastic Petri nets; the software testing resource sharing model is created using Petri nets' synchronization conflicts; shared resource scheduling and selecting options are designed using the immediate transition and tokens in a shared location. Based on this, the shared resource software testing task scheduling and selecting model that includes n types of tasks and m testing resources is created. It gives four scheduling options and three selecting options and twelve combinations from them. We use self-developed Petri nets based task scheduling and selecting simulation program to simulate a practical situation, compare different behaviors and performances of different options and analyze the causes.

References

[1] J. Song, Software Testing Technology and Developing an Automated Software Testing Tool[D]. Beijing University of Posts and Telecommunications, 2007 (in Chinese)

[2] C. Yang, Research and Implement of GUI Based Auto-testing Architecture[D]. Donghua University, 2008 (in Chinese)

[3] Murata T. Petri nets: Properties, analysis and applications[J]. *Proceedings of the IEEE*, 1989: 541-580.

[4] M. Pan, W. Wu. Performance analysis of scheduling rules in remanufacturing operations using stochastic Petri nets[C]. *Networking, Sensing and Control (ICNSC), 2014 IEEE 11th International Conference on. IEEE*, 2014:120 - 125.

[5] Wolff R W. Poisson Arrivals See Time Averages[J]. *Operations Research*, 1982: 223-231.

[6] Gross D. Fundamentals of queueing theory : solutions manual to accompany[M]. Wiley, 2008.

[7] Marsan M A, Balbo G, Conte G, et al. Modeling with generalized stochastic Petri nets[J]. Acm Sigmetrics Performance Evaluation Review, 1998, 26.

[8] C. Lin. Stochastic Petri nets and Performance Evaluation of System. Beijing: Tsinghua University Press, 2005, chapter 7. (in Chinese)