DISTRIBUTED T-WAY TEST SUITE GENERATOR SYSTEM USING MASTER WORKER EXECUTION MODEL

Ong Hui Yeh, Kamal Zuhairi Zamli

School of Electrical and Electronic Engineering
Universiti Sains Malaysia, Engineering Campus
14300 Nibong Tebal, Seberang Perai Selatan,
Pulau Pinang, Malaysia
Tel: +604-5995096

Email: yeh_1985@hotmail.com, eekamal@eng.usm.my

Abstract

Advancement of software is always in line with its capacity. When the software is large, considering all exhaustive testing is impossible. Sequentially implemented t-way testing strategies have been reported of its efficiency in the past literature to address the aforementioned issues; however, they suffer limited memory and retarded performance. Distributed computing appears to be a good avenue in this case. In this paper, we propose the implementation of twayGenerator (our previous work of sequential t-way testing strategy) in a distributed system by using Master Worker execution model. The speedup performance is the main concern in the research. It is generally predicted the degree of increment of speedup performance could be improved based on the number of computers used to run the test cases generation. The expected outcomes and practical results have been cited out based on the supportive discussions.

Keywords: *software testing, interaction testing, distributed computing, Master Worker execution model, t-way testing.*

I. INTRODUCTION

Software testing can be regarded as any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. It can also be defined as the process of executing a program or system with the intent of finding errors [1]. Software testing can be considered as one of the most important activities since it covers as much as 40 to 50 percent of total software development costs [2] [3]. It measures the reliability of a typical software system whereas this aspect has become an important factor especially when software is employed in harsh, life threatening or critical (safety) applications such as airplane control systems and biomedical instrumental devices. Rigorous software testing is therefore needed since lacking of testing often leads to disastrous consequences including loss of data, fortunes and even lives [4].

As the increments of size and capacity of software in line with its advancement and complexity, many combinations of possible inputs, parameters, hardware and software environments, and system conditions need to be tested and verified. However, it is impossible to consider all exhaustive testing. The main factors that inhibit such consideration include resource constraints and costing factors as well as strict time to meet market deadlines. Thus, there is a need for a systematic strategy in order to reduce the test data set into manageable ones [5].

Interaction testing (t-way testing) is an effective approach to address aforementioned issues, where t is the level of interaction among the system's input parameters, may be set to 2, 3, 4 or higher [6]. In t-way testing, a set of test cases is generated to cover a subset of the possible combinations of the system's input parameters, rather than trying to cover all possible combinations. The rationale for t-way testing stemmed from the fact that from empirical observation, the number of input variables involved in software failures is relatively small (i.e. in the order of 2 to 6), in some classes of software.

Numerous strategies have been developed in the past literature on t-way testing but they are more focused on pairwise (where t=2) testing [3] [7] [8] [9] [10] [11]. As an example, Lei and Tai have proposed a test generation strategy for pairwise testing, which is know as In-Parameter-Order (IPO) [8]. Given a system with two or more input parameters, the IPO strategy create a pairwise test set for the first two parameters then extends the test set to produce a pairwise test set for the first three parameters, and proceed to do so for each additional parameter. Earlier literatures claimed that pairwise testing (where t=2) can be effectively to disclose most faults in particular software system [12]. Nevertheless, such approach cannot be generalized to all software system faults especially when there are significant interactions between parameters.

Herein, recent attentions have been focused more on t-way testing which could cope with the interaction strength greater than two (t≥2). Several strategies have been reported, such as Automatic Efficient Test Generator (AETG) system [13] and Test Vector Generator (TVG) tool [14]. For instant, AETG system utilized a new approach to testing that uses combinatorial designs to generate test cases that cover t-way combinations of a system's test parameters [13]. On the other hands, J. Arshem then proposed Test Vector Generator (TVG)

tool based on the extension of AETG strategy to support t-way testing [14]. Even though these strategies are reported to be efficient as far as the number of test cases generated is concerned, most of them are sequentially implemented, which might be supportive for a system with limited number of inputs. With the rapid growth of software nowadays, they might encounter the bottom neck of insufficient memory and suffer with the retarded performance when there are massive of input variables to be taken into account. As a consequence, distributed system implementation appears to be a good avenue in this case.

This paper proposes a distributed t-way test suite generator system by using Master Worker execution model. Specifically, a non-deterministic t-way strategy, twayGenerator [15] has been adopted in the implementation. Basically, the twayGenerator will be executed on multiple machines instead of single computer by utilizing distributed computing techniques. Nonetheless, several issues are to be concerned. For example, it is crucial to apply the load balancing algorithm for distributed shared memory processor. Next, the distributed program often deals with heterogeneous environments, network links of varying latencies, and unpredictable failures in the network or the computers. Therefore, the algorithm shall be designed to be scalable, portable and easy-to-use.

This paper is structured as follows: Section I illustrates the introduction of t-way testing whereas Section II describes the implementation of distributed twayGenerator. Section III gives the expectation outcomes. Section IV shows the practical results. Section V gives the conclusion of this work and cites the possible future work to be carried on.

II. METHODOLOGY

Before we further discuss the implementation of Master Worker execution model in distributed twayGenerator, there is an issue arises to be emphasized, i.e. the problem of distributing workloads on multiprocessor in a network. Adequate fair dynamic

This research is funded and supported by Universiti Sains Malaysia Research University Postgraduate Research Grant Scheme (USM-RU-PGRS, Grant Number: 1001/PELECT/8033001) and USM postgraduate fellowship scheme.

load distribution techniques would be required to efficiently running parallel applications on distributed computing environments. This is because the unfairly distributions techniques may cause certain processors overloaded while other processors under loaded and waste its bandwidth to run jobs. The whole system wastes time on waiting jobs that are queued on overload processor while nothing done by other processors. Load balancing is therefore needed as it is usually achieved by relocating application tasks from busy nodes to lightly loaded or idle nodes [16]. It is one of the most challenging issues in attaining high performance in heterogeneous systems. Scheduling algorithms were devised to perform distributing loads from server to processors. Thus, high performance load balancing algorithm is an algorithm that could distribute loads fairly to processors and prevent overloaded condition.

In order to employ distributed computing technique on distributed twayGenerator, the idea on Master Worker (MW) execution model (also known as Master Slave model) has been proposed. In this model, there are two distinct types of processes: Master and Workers. Master decomposes the problem into small tasks; distribute to Workers and correlating their output into a global results whereas Workers typically collect and execute the tasks and then sending back to the Master. It is proved that the Master Worker execution is efficient in developing applications with different degrees of granularity of parallelism (grain size) and is particularly useful when the dependencies between tasks are low [17]. This is favor to distributed twayGenerator as the distributing tasks are self-regulating from one another.

Furthermore, daemon is integrated with our proposed strategy. The word "daemon" actually comes from the Greek language, meaning an "inner or attendant spirit" (Oxford American Dictionary). This is a fitting name, as a computer daemon is a constantly running program that triggers actions when it receives certain input. For example, a printer daemon spools information to a printer when a user decides to print a document. A daemon running on a mail server routes incoming mail to the appropriate mailboxes. Web servers use a Hyper Text Transfer Protocol Daemon (HTTPD) that sends data to users when they access Web pages [18].

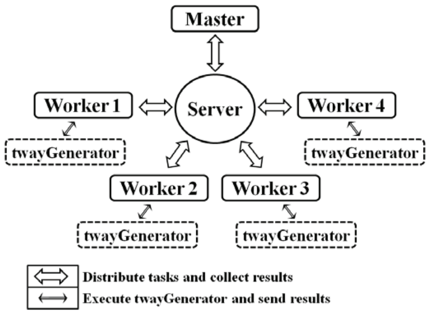

Fig. 1: The Customized Master Worker Execution Model

Based on distributed twayGenerator, we have modified and customized the Master Worker execution model as depicted in Fig 1. The further details as shown below:

Master

Firstly, the Master receives the inputs specifications (parameters, values and t-way) from the user. Based on that, the Master will manipulate in order to produce all the possible combinations of binary numbers.

For example, giving the inputs which consisting of 5-valued parameter, 2-valued variables and 4-valued way of interactions. Adopting the twayGenerator algorithm, the Master then generates the following combinations of binary numbers: 01111, 10111, 11011, 11101 and 11110. We term each of these binary numbers as a "thread". Then, the Master gets the status of the available Workers via tuple space (server) that has been created. Presume that four Workers are available in our discussion here.

The Master starts to distribute the tasks to the available Workers by sending the tuples which each consisting of a binary number (thread) and selected Worker's Internet Protocol (IP) address to appointed Workers respectively. The processes for the Master to allocate the threads to the Workers are known as multithreading. After the multithreading process, the Master waits for the replies from the Workers which indicating the completion of the given tasks.

There are still left of one unprocessed binary number. It will be assigned to a Worker that has completed the first task faster than the residues. Note that the results (generated tests sets) from the Workers, some are repetitions; thus, eliminating the repetitions are required. Hence, Master will filter out the results (from the Workers) to generate a final output (i.e., test sets without repetitive).

Worker Process

The Workers initialize their tasks by sending their status into tuple space (to show their availability). By using the matching algorithm, the Workers scan on tuple space to detect their own tasks. The working of this algorithm is based on the unique IP address for each node from the Master and the four Workers.

Daemon is then integrated into the Worker process as it will trigger out the execution of another program when accepting certain key words from the received tuple. In short, a program which consisting of t-way algorithm (twayGenerator) is initialized and executed based on the binary number obtained from the Master. Upon the completion of executing the program, the results are sending into tuple space. Workers are then turning into ready mode again for incoming tuple if any. The customized MW model of the example mentioned above is illustrated at Fig 1.

Based on the proposed strategy, several assumptions would be made here. First, the tuple space server has to been initiated before the user enters the inputs to the Master. Next, we presume that there is no occurrence of network congestion. Congestion or disconnecting of networking may cause the server error. All the approaches and methods as described in this section can be implemented via Java programming language attached with International Business Machines, IBM tuple space library. This research proposed the designation and implementation of load balancing algorithm on twayGenerator in real network. Next section stating the expected outcomes for distributed twayGenerator.

III. EXPECTED RESULTS

In this section, we propose the expected results as the research is in the progression. Speedup performance is the major aspect of our research. Ideally, the degree of growth of this feature is predicted to be directly proportional to the number of computers been used to run the test data sets generation.

In ideal case, it could be stated that the time required to complete the tasks is expecting to reduce by half if two computers are utilized in generating the test set. Then, the same goes to getting one third required time by using three computers and so on. Hence we could generalize that the speedup feature can be improved by **N** factors (reducing the time required by **N** times) if there are **N** units of computers have been used. These could be clear shown at Fig 2. Notes that a in the graph is representing the portion of time required for the single computer to run a specific test sets generation. It is forecasting that we need **a/N** of time to run the same program by using **N** units of computers for ideal case.

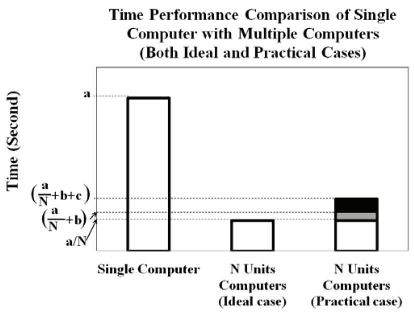

Fig 2: Expected Results of Distributed twayGenerator

However, all these aforementioned deductions are hard to attain as certain practical considerations have been excluded. In the practical case, time required for multithreading process has to be taken into account as well. That is the time consumed for the Master to distribute the threads to the Workers as well as the moments of the Workers spent on sending the results to the Master. The amount of time, termed as b is denoted as the grey colored region in Fig 2.

The effect of value **b** could be ignored if compared with the value of **a**, provided that the time for generating the test sets by single computer is long as shown at Fig 2. Nevertheless, there is **a** tradeoff of implementing our approach. This can be described as if it is requested to generate the smaller number of test sets (with less parameters and variables, saying 5 parameters 2 variables combinations), then give rise to the smaller value of a so that it is smaller or equal to the value of **b**. In this case, it shows that our strategy consumes more time on multithreading process rather than generating test sets. Thus, single computer is said to perform better for fewer test sets. However, our aim is to deal with mass volume of computational tasks derived from the combinations of large numbers on parameters and variables. The performance regarding to lesser test sets shall be negligible.

There is another important factor worth to be mentioned. The speedup gain is actually also dependent on the fraction of an algorithm that could be parallelize and executed into distributed system. However, rare distributed software could be solely executed in parallel. Therefore, twayGenerator also hinders its speedup performance by the fraction of sequential fraction of the algorithm, denoted as **c** (the black colored area)

Table 1. Generation Time (in second, s) and Speedup Gain for 5 to 10 5-Valued Parameters of 4-way (t=4) Testing

Sequential twayGenerator								
# parameters			**5**	**6**	**7**	**8**	**9**	**10**
time , s			1.937	5.328	12.328	24.453	43.453	73.313
Distributed twayGenerator								
# parameters			**5**	**6**	**7**	**8**	**9**	**10**
# workers	**1**	**time, s**	3.079	8.969	20.797	41.406	70.531	121.719
		speedup gain	0.629	0.594	0.593	0.591	0.616	0.602
	2	**time, s**	1.906	4.937	10.922	21.333	38.828	64.219
		speedup gain	1.016	1.079	1.129	1.146	1.119	1.142
	3	**time, s**	1.468	3.688	8.469	16.703	29.547	49.672
		speedup gain	1.319	1.445	1.456	1.464	1.471	1.476
	4	**time, s**	1.453	3.250	7.172	14.188	25.094	42.157
		speedup gain	1.333	1.639	1.719	1.723	1.732	1.739
	5	**time, s**	1.109	2.875	6.500	12.781	23.172	38.234
		speedup gain	1.747	1.853	1.897	1.913	1.875	1.917
	6	**time, s**	1.109	3.000	6.484	12.813	22.781	38.016
		speedup gain	1.747	1.776	1.901	1.908	1.907	1.928
number of test cases generated			3,125	9,375	21,875	43,750	78,750	131,250

in Fig 2. Nonetheless, this portion is predicted to be miniature as distributed twayGenerator could be distribute most of the computational loads via Master Worker execution model.

According to our proper and supportive justifications based on several important practical features, it is predicted our strategy is capable to increase the speed up performance as illustrated at Fig 2.

IV. PRACTICAL RESULTS

In this section, the practical results of distributed twayGenerator will be discussed. In this case, the generation time (practical results) is referring to the time required for master to distribute tasks and workers generate the test cases. Note that, the time required for master to remove the repetitive test cases (as discussed in Section II) are excluded as this research is under progression. In order to demonstrate the effectiveness of distributed twayGenerator against sequential twayGenerator, there are 5 to 10 parameters (where each of which is 5-valued) have been executed. In this case, t is set to 4 and the number of workers is varying from 1 to 6. Table 1 shows the results in terms of generation time and the speedup gain. Here, speed up is defined as the ratio of the time taken by sequential twayGenerator to the time taken by distributed tway Generator.

By referring to Table 1, the number of test cases is increased as the number of parameters increases from 5 to 10. This is also in line with the time to generate the test cases (i.e., longer time is needed to generate more test cases) for each particular number of workers employed. Besides that, the test cases will be generated faster with the higher number of workers employed for a typical input. As far as generation time is concerned, distributed twayGenerator shows appreciable speedup gain against sequential twayGenerator. It gives higher speedup gain with more workers are employed. However, an exceptional issue has been discovered. The undesired speedup gains (where the gain less than unity) were obtained when only one worker is employed in the system. In general, the distributed twayGenerator improves the time performance from sequential twayGenerator. Though, it exhibits non-ideal speedup gain as pre-described in previous section.

V. CONCLUSION

To conclude, sequential approach t-way strategies will encounter the conflicts of overloaded memory as the combinatorial testing are always deal with heavy computational tasks in generating test sets. Withstanding of that, the constraint from the aspect of limited memories in the single computer could be solved by utilizing distributed computing technique. This paper proposed an implementation strategy of distributed twayGenerator by using Master Worker execution model. Based on the expected outcomes and the practical results that accomplished with reasonable discussions been stated, Master Worker execution model could be favorable to substitute the sequential execution in order to support higher loads of inputs in t-way testing. As this research is under progress, more experimental data and analysis would be justified in the future work. It is hypothesized that our strategy would contribute to the positive impacts on current research of t-way testing by further enhancing the test data generation time performance of twayGenerator.

ACKNOWLEDGMENT

The authors appreciate the ideas and concepts provided by Mohammed I. Younis.

REFERENCES

[1] J. Pan, "Software Testing (18-849b Dependable Embedded Systems),"

Topics in Dependable Embedded Systems, Electrical and Computer Engineering Department, Carnegie Mellon University, 1999.

[2] Y. Cui, L. Li and S. Yao, "A New Strategy for Pairwise Test Case Generation," in IITA 2009: *Third International Symposium on Intelligent Information Technology Application*, 2009, vol.3, , 21-22 Nov. 2009, pp.303-306.

[3] J. Gao and Y. Hu, "A Novel Generation Algorithm of Pair-Wise Testing Cases," in PRDC '09:*15th IEEE Pacific Rim International Symposium on Dependable Computing*, 2009, 16-18 Nov. 2009, pp.43-48.

[4] M. I. Younis, K. Z. Zamli, and N. A. M. Isa, "Generating Pairwise Combinatorial Test Set Using Artificial Parameters and Values," in ITSim 2008: *International Symposium on Information Technology 2008, pp. 1-8.*

[5] M. I. Younis, K. Z. Zamli, and N. A. M. Isa, "Algebraic Strategy to Generate Pairwise Test Set for Prime Number Parameters and Variables," in ITSim 2008: *International Symposium on Information Technology* 2008, pp. 1-4.

[6] P. J. Schroeder, P. Bolaki and V. Gopu, "Comparing the fault detection effectiveness of n-way and random test suites," in ISESE '04: Proceedings. *International Symposium on Empirical Software Engineering*, 2004, 19-20 Aug. 2004, pp. 49- 59.

[7] R. Abdullah, M. F. J. Klaib, K. Z. Zamli, N. A. M. Isa, and M. I. Younis, "G2Way-A Backtracking Strategy for Pairwise Test Data Generation," in APSEC '08: *15th IEEE Asia-Pacific Software Engineering Conference*, 2008. Beijing, China, 03-05 December 2008, pp. 463-470.

[8] K. C. Tai, Y. Lei, "A test generation strategy for pairwise testing," in *IEEE Transactions on Software Engineering*, vol.28, no.1, Jan 2002, pp.109-111.

[9] J. D. McCaffrey, "Generation of pairwise test sets using a simulated bee colony algorithm," in *IEEE International Conference on Information Reuse & Integration 2009*: IRI '09, 10-12 Aug. 2009, pp.115-119.

[10] J. D. McCaffrey, "Generation of Pairwise Test Sets Using a Genetic Algorithm," in *33rd Annual IEEE International Computer Software and Applications Conference 2009*: COMPSAC '09, vol.1, 20-24 July 2009, pp.626-631.

[11] M. I. Younis, K. Z. Zamli, and N. A. M. Isa, "IRPS-An Efficient Test Data Generation Strategy for Pairwise Testing." in *12th International Conference on Knowledge-Based Intelligent Information & Engineering Systems: KES 2008*, Zagreb, Croatia, Springer-Verlag, Berlin, Heidelberg, pp. 493-500.

[12] W. B. Mugridge, D. M. Cohen, P. B. Gibbons, and C. J. Colbourn, "Constructing Test Suites for Interaction Testing," in *25th International Conference on Software Engineering, Portland, Oregon, USA*, 2003, pp. 38-48.

[13] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design," in *IEEE Transactions on Software Engineering* 23(7), July 1997, pp. 437-444.

[14] J. Arshem, Test Vector Generator Tool (TVG), Available from http://sourceforge.net/projects/tvg/, last accessed on December, 2009.

[15] K. Z. Zamli, "Non-Deterministic T-Way Strategy for Systematic Combinatorial Test Data Reduction," in MySEC'08: *The 4th Malaysian Software Engineering Conference: Empowering Software towards the Development of Human Capital*, Kuala Terengganu, Malaysia, 16-17 December 2008.

[16] P. Yi, and T. Y. Laurence, *Parallel and distributed scientific and engineering computing*, Nova Science Publishers, Inc. US, 2004.

[17] C. Gabriela, and I. Daniel, *Scientific computing in electrical engineering*, Springer, Berlin, Heidelberg, New York, 2007, pp. 412.

[18] Daemon. (2009). *The Tech Terms Computer Dictionary*. Retrieved from: http://www.techterms.com/definition/daemon, last accessed on Feb, 2010