Int. J. Advance Soft Compu. Appl, Vol. 5, No. 3, December 2013
ISSN 2074-8523; Copyright © SCRG Publication, 2013

A Distributed T-Way Test Suite Generation Using "One-Parameter-at-a-Time" Approach

Zainal H. C. Soh, Syahrul A. C. Abdullah, and K. Z. Zamli

Faculty of Electrical Engineering, UiTM Pulau Pinang, Penang, Malaysia
e-mail: zainal872@ppinang.uitm.edu.my

Faculty of Electrical Engineering, UiTM Shah Alam, Selangor, Malaysia
e-mail: bekabox181343@salam.uitm.edu.my

Faculty of Computer Systems & Software Engineering, UMP, Pahang, Malaysia
e-mail: kamalz@ump.edu.my

Abstract

This paper presents a new distributed test suite generation for t-way testing, called TS_OP, using Map and Reduce software framework based on tuple space technology environment. TS_OP takes a one-parameter-at-a-time strategy and is capable of supporting high interaction strength (i.e. t>5). Internally, TS_OP coordinates and distributes the test case generation workload amongst participating workstations. An encouraging result is obtained from experimentation on the optimality of test suite size generated and on the speedup gain in multiple machine environments. Benchmarking studies in term of size of generated test suite against existing parameter based strategies (i.e. IPOG, MIPOG, IPOG-D, IPOG-F and IPOG-F2) indicate that TS_OP gives competitive results.

Keywords: *T-way Testing, Combinatorial Interaction Testing, Distributed Shared Memory, Map and Reduce, Tuple Space Technology, Space Based Remoting*

1 Introduction

Nowadays, most of available *t*-way (where *t* indicates the interaction strength) testing strategies for software interaction testing were implemented on a standalone machine and cannot be extend to work on multiple machine environments. However, the computational power and memory space of a standalone workstation appear to be insufficient when dealing with large input parameter and high interaction strength. In this case, the test generation time can potentially take days to complete due to high volume of data and intense computational works.

Moreover, for large and complex real software system, the sampling search space as well as the interaction between parameter values of higher order *t*-way is likely to be huge. The resultant number of test case in test suite also increases exponentially as the value of interaction strength, *t* increases. Therefore, considering a higher order *t*-way test suite generation for large input parameter potentially require high computational power and large memory space.

In order to address the aforementioned issues, a new distributed *t*-way test suite generation strategy is been developed, called TS_OP, capable of distributing the computing work among network of participating workstations. Here, high level APIs for the Map and Reduce mechanism on Tuple Space middleware (i.e. GigaSpaces) are employed to distribute the computing work using the tuple space as a transport layer. While running in multiple machines environments, the memory resources from network of workstations are combined so as to garner the computational power from the connected workstations to perform the required computations.

This paper presents a distributed *t*-way test suite generation using Tuple Space technology. The remainder of the paper is organized as follows. Section 2 gives some insight on recently published related works on test suite generation strategies. Section 3 describes the design and implementation approach for a distributed TS-OP on single machine environment and multi machine environment. Section 4 discusses the experimental result of the developed strategy in term of speedup gain and comparison with other existing strategies. Finally, section 5 draws our conclusions and point out the ideas for future extension of this work.

2 Related Work

In line with the scope of this paper, we review the existing literature based on the pure computational approach. For pure computational algorithm approach, the *t*-way test suite generation strategy can be divided into two main categories known as “one-test-at-a-time” strategy and “one-parameter-at-a-time” strategy. As for “one-parameter-at-time” strategy, the main example is IPOG[1] which support *t*-way testing. In this strategy, a *t*-way test suite for the first *t* parameters is generated, and then extends each test case in the test suite to cover for the first *t*+1 parameters in horizontal extension phase. All the test cases in the test suite are extended with a new parameter value that covers maximum uncovered interaction elements. IPOG utilized the greedy search technique to calculate maximum interaction element combinations coverage. After all test cases are extended and there are still uncovered interaction elements then the vertical extension is required. In vertical extension, a new test case is added into the test suite to cover for the uncovered interaction element combination. After the entire interaction elements are covered, the vertical extension is completed. The partially built test suite is extended by adding a new parameter until all parameter in the system are covered.

A number of variants have also been developed to improve the IPOG's performance within ACTS tool [2-4]. These variants including: IPOG-D [5], IPOG-F[6], IPOG-F2[6], MIPOG, G_MIPOG and MC-MIPOG. IPOG-D is a deterministic strategy that combines the IPOG strategy with a recursive D construction to minimize the search space during generation of the test suites. Although IPOG-D can generate the test suite faster than IPOG, their test size usually bigger than IPOG. Both IPOG-F and IPOG-F2 are non-deterministic strategies with faster test generation time compared to IPOG. In general the size of test suite generated by both strategies is competitive as compared to IPOG. Although both strategies can support uniform and mixed input parameter setting, the test size optimality seem do not extend to the mixed input parameter value and IPOG seems to do better with these situation. Unlike IPOG-F, IPOG-F2 is implemented with a heuristic search for horizontal growth algorithm thus permits faster test generation time as compared to IPOG-F.

MIPOG strategy is a deterministic strategy (i.e. each runs will produce a same test suite size. Unlike IPOG, in horizontal extension, the MIPOG strategy optimizes the extended test case by selecting a value that covers the maximum number of uncovered interaction element combinations. Also, MIP*OG optimizes the don't care value by means of searching of uncovered interaction element that can be combined with this test case during horizontal extension. In vertical extension, MIPOG creates a new test case by searching for a combination of interaction elements that covered the most uncovered *t*-way combinations. This step, while improving the test suite size, also increases the overall test generation time of MIPOG. Both G_MIPOG and MC-MIPOG are built based on sequential MIPOG strategy with parallel processing of the test suite generation work. G_MIPOG is implemented on a grid network while MC-MIPOG is run on a multicore system. Both strategies support higher order of *t* for test suite generation and can produce a smaller test suite as compare to others variant of IPOG.

Another category is “one-test-at-a-time” strategy, the typical example is Automatic Efficient Test Generator (AETG) which builds a test suite using greedy search technique to generate a complete test case iteratively until all the interaction element combinations are covered. AETG produce a non-deterministic test suite solution due to their random selection of next parameter order. On the other hand, it does produce a quite competitive and optimal number of test cases due to selection of the best test case from selection of 50 candidate test cases in one test case generation iteration.

Others example of strategy that lie in the same group as AETG are TCG[7], Dens, ITCH[8], TVG[9], PICT[10] and Jenny[11]. In TCG, a fixed rule of test case selection is used, thus produce a deterministic solution. In Dens, Bryce et al develop a higher strength test suite generation using greedy algorithm to select the parameter value based on their density value. Due to random insertion of first value into the test case, the test case generated was no longer deterministic. ITCH is an IBM’s Intelligent Test Case Handler that uses a combinatorial approach based on exhaustive search to generate the test case and required a substantial time to complete. As for TVG and Jenny, both support *t*-way testing but there are only limited information regarding both strategy implementations in written literature. Both tools can be downloaded and provide an executable source code for free at their respective websites.

Finally, in their work, Younis et al[12] demonstrates that MC-MIPOG performs better than other t-ways testing tool that based on “one-test-at-a-time” strategy, both in terms of test generations time and the sizes of the generated test suites. For these reason we have adopted the “one-parameter-at-time” strategy as our basis of design approach with distributed test suite generation across a multiple machines environments and MC-MIPOG itself as our benchmark.

3 Distributed TS_OP Design

In this section, the design of distributed *t*-way test suite generation strategy based on “one-parameter-at-a-time” strategy called TS-OP strategy is explained. The strategy is implemented using Map and Reduce mechanism on network of workstations using Tuple Space technology middleware known as GigaSpaces XAP 8.0.[13] In this strategy, a master process is called TS_OP Feeder Processing Unit (PU) and the worker process is called TS_OP Processor PU.

Firstly, an application model of *t*-way test suite generation is designed and implemented comprising of TS_OP Feeder PU and TS_OP Processor PU as shown in Fig. 1. The Feeder PU is responsible to feed all data into each partition space using the *Input Data Loader* services and controls test suite generation as well as delegates the final TS using *Test Data Feeder* services. The TS_OP Processor PU is used to generate the test case and calculate the interaction coverage of generated test case by using *IECProcessor* services. All the services are loosely connected to all the data in space (i.e. *t*, *vi*, *ParmSet* and *ieSet*). All the PUs are wired together using a Spring configuration file, *pu.xml*.

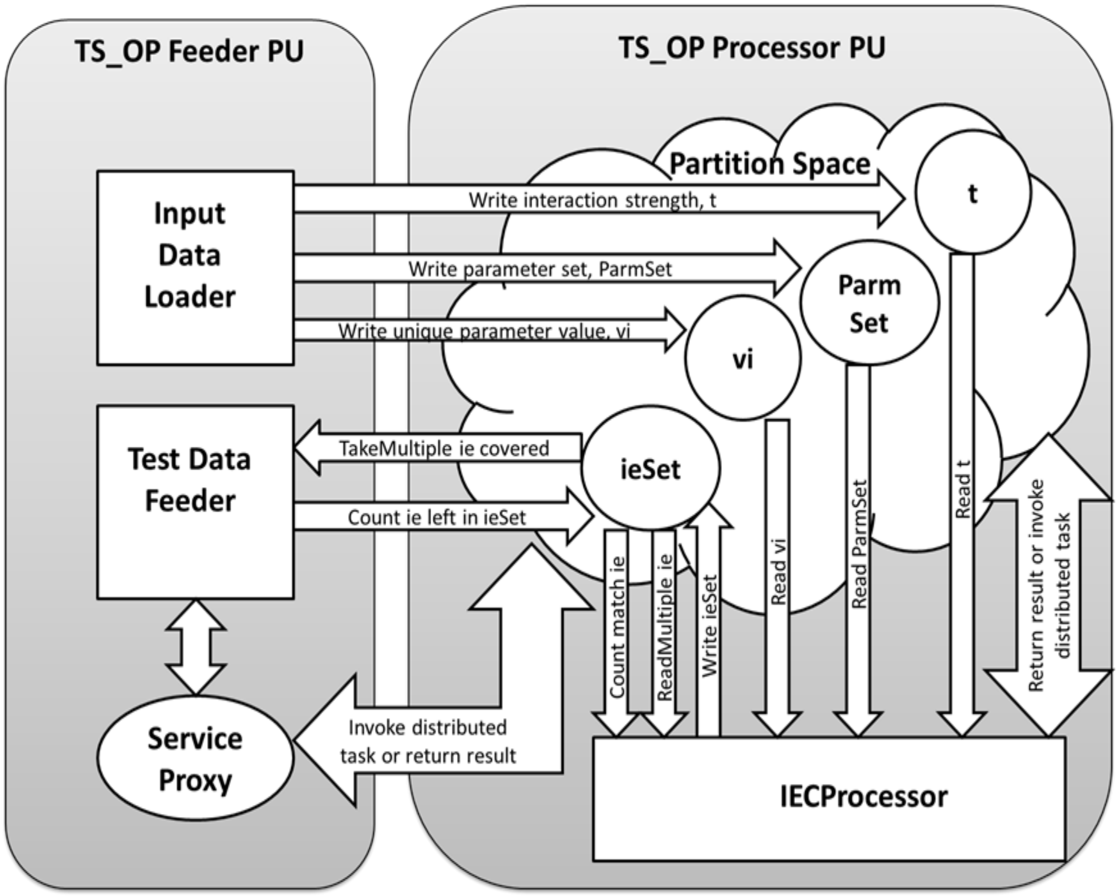

Fig. 1: A model of one TS_OP Feeder PU and one TS_OP Processor PU.

Using above single machine model, a distributed TS_OP strategy is designed and implemented. The first step in distributed TS_OP strategy is to identify the number of worker process or TS_OP Processor PU. In TS_OP strategy, the number of worker process or TS_OP Processor PU depends on the highest number of parameter value among given input parameter.

Next step is to identify the distributed data and the common data at each partition space. The distributed data in TS_OP strategy on each partition space is a one uniquely assigned input parameter value, *vi*. This unique value *vi* is used to generate the corresponding *ieSet* for that partition space. Hence, the interaction elements exist in each partition are ensured to be different from others partition space resulting from the assigned parameter value. The common data in all partition spaces are the interaction strength, *t* and the input parameter, *ParmSet.* All data is constructed as Plain Old Java Object (POJO) data and stored in all the partition space.

The third step in distributing the TS_OP strategy is to identify all tasks running on both TS_OP Feeder PU and TS_OP Processor PU. All task services running on the TS_OP Feeder PU are considered as a master task whereas all tasks running on TS_OP Processor PU are considered as a distributed task. All master tasks that are running on the TS_OP Feeder PU are preload of the interaction strength, *t*, the input parameter; *ParmSet* and the unique value, *vi* to all partition space, generation of the *t*-way test suite for the first *t* parameter, control of test generation and remotely execution of all distributed task on TS_OP Processor PU, selection and storage of selected test case in final test suite, *TS*, deletion of the interaction element covered by selected test case in all partition space, removal of redundant test case after a vertical extension and display and storage of the final test suite into a log file.

The distributed task services are the tasks that have been remotely executed by TS_OP Feeder PU and interact only with distributed data at their respective partition spaces. The synchronous mode of space based remoting service, called Map and Reduce mechanism, is utilized by TS_OP Feeder PU to simultaneously invoke the distributed task service on all TS_OP Processor PU. A distributed processing is achieved while running in this mode. Five tasks are running on TS_OP Processor as distributed tasks:-

(i) Generation of the interaction element data using assigned unique value, *vi*.
(ii) Generation of the extended test case by inserting the assigned unique value, *vi* to that test case.
(iii) Generation of the extended test case with don’t care by merging with possible interaction element combinations
(iv) Generation of the complete test case by merging between possible interaction element combinations
(v) Calculation of the maximum interaction coverage value for the test case generated, selection of test case with highest value of maximum interaction coverage and return selected test case with maximum interaction coverage to TS_OP Feeder PU via a Reducer.

Finally, the distributed TS_OP strategy is deployed into network of workstations with distributed data. The TS_OP strategy is implemented on partitioned topology using the GigaSpaces Service Grid. For a single machine environment, the TS_OP Feeder PU and all TS_OP Processor PU with their collocated partition space run on different GSC within a same machine. While for multiple machine environments, TS_OP Feeder PU and each TS_OP Processor PU with their respective partition spaces are distributed across several different physical machines in different GSC. Here, the multiple machine environments are capable to harness the combined memory resources and computing power of connected workstations.

As for multiple machine environments as shown in Figure 2, the TS_OP Feeder PU has two collocated services, *Input Data Loader* services and *Test Data Feeder* services in GSC 4 on machine 4. The complete algorithm for the services running in TS_OP Feeder PU is shown in Fig. 3. As for all three TS_OP Processors PUs, their collocated partition spaces are running on machine 1, 2 and 3 respectively. Summing up, the complete algorithm for the *IECProcessor* service running in TS_OP Processor PU is shown in Fig. 4.

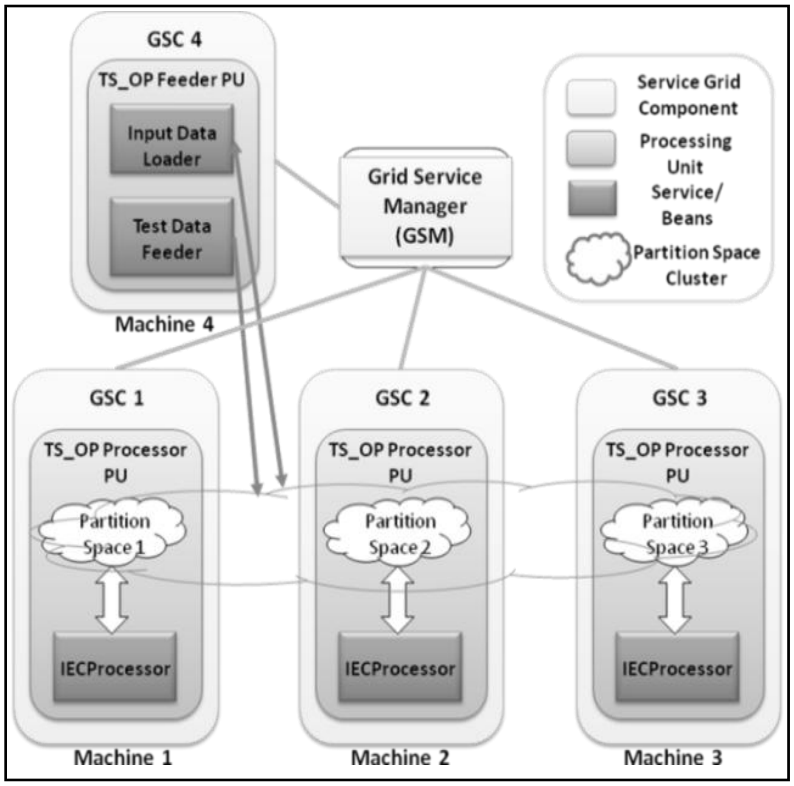

Fig. 2: Grid Service Manager managing 4 Grid Service Containers for multiple machine environments with 1 TS_OP Feeder PU and 3 TS_OP Processor PU.

```
Algorithm TS_OP Feeder PU( t, ParmSet)
begin
1. initialize test suite TS to be an empty set;
2. denote the parameters in ParmSet, as P1, …, and Pn;
3. send ParmSet and t to GigaSpaces;
4. assign unique values, vi to each TS_OP Processors;
  { for the first t parameters }
5. add into TS a exhaustive test case for first t parameters;
6. for  parameter Pi, i =t+1, …,n do
    begin
      { horizontal extension for parameter Pi }
7.      send command generateIE to all TS_OP Processor
        using space based remoting and stored in ieSet;
8.        for each test τ = (v1, v2, …, vi-1) in test suite TS do
9.            send command construct test case and τ  to all
              TS_OP processor using space based remoting;
10.           wait for all TS_OP Processor send τ′ with maxIE;
11.            reducer choose the τ′ with highest maxIE;
12.            add selected test case  τ′ to TS;
13.            delete all covered interaction element in ieSet ;
        { vertical extension for parameter Pi }
14.       while (ieSet is not empty) do
15.            send command calculateIECMax to all TS_OP
               Processor to merge possible interaction element
               combination into tsm;
16.            reducer choose the tsm with highest maxIE;
17.            add selected test case  tsm to TS′;
18.            delete all covered interaction element in ieSet ;
19.       remove redundant tsm in TS′
20.  add temporary TS′ to TS
    end
21.  return TS;
end
```

Fig. 3: TS_OP Feeder PU Algorithm.

```
Algorithm  TS_OP Processors PU
begin
1. initialize as one PU with dedicated partition space;
2. read t, ParmSet from their dedicated partition space;
3. read assigned value, vi on their partition space;
4. for  parameter Pi, i =3,4,…,n do
      { horizontal extension for parameter Pi=v }
5.    if receive command generateIE and Pi from TS_OP
         Feeder through space based remoting;
6.      generate t-way ie between Pi and{P1…Pt} using
        assigned parameter value and stored into in ieSet
        TS_OP Processors partition;
7.      let ieSet be the set of all pair combinations of values
          between Pi and each  of P1,  P2,…,Pi-1;
8.    if receive command construct test case and τ from
         TS_OP Feeder through space based remoting;
9.      if (τ  not contains don't care)
10.        insert assigned value (v) into τ;
11.        calculate the maxIE;
12.        send τ′ and maxIE to TS_OP Feeder via reducer;
13.   else  merge τ with possible interaction element
            combination in ieSet into τo ;
14.         calculate the maxIE;
15.        send τo and maxIE to TS_OP Feeder via reducer;
      { vertical extension for parameter Pi }
16.   if receive command calculate IEC Max from TS_OP
        Feeder through space based remoting;
17.     while (ieSet is not empty) do
18.          merge possible interaction element
             combination in ieSet into tsm ;
19.          calculate the maxIE;
20.      send tsm and maxIE to TS_OP Feeder via reducer;
  end
```

Fig. 4: TS_OP Processor PU Algorithm

4 Evaluation

To evaluate the performances of the developed strategy, a number of experimentations were undertaken. Here, our evaluation has two main aims. Firstly, we want to investigate the scalability performance of distributed strategy in term of speedup and test size ratio between single machine and multiple machine environments. Secondly, a comparison in term of the generated test suite size of the developed distributed strategy against existing parameter based strategies.

4.1 Scalability analysis on speedup

To measure the speedup gain of developed distributed strategy, network of 5 identical and homogeneous workstations with the same operating system (Window XP), processing power (HP PC Pentium Core 2 2.13GHz) and main memory capacity

(4GB of RAM) interconnected using a 2950 Cisco switch. The network of workstation is implemented on a star topology with the GigaSpaces middleware running on each workstation. The input parameter setting is uniform input parameter of fixed parameter, *p*=10 with parameter value, *v*=5, 5^{10} and interaction strength of *t*=4.All of these experiments were carried out on single and multiple machine environments ranging from 2 to 5 machine. All results of the test size and test generation time between each machine environment are recorded in Table 1. Here, test size ratio is used to identify the variation in test suite size for each machine environment. The test size ratio is deduced by dividing the test size on their current machine setting with the test size on maximum deployable machines.

Table 1: The test size ratio and speedup for *v*=5 and *p*=10 with interaction strength *t*=4 on five different machines.

Number of Machines	**TS_OP**			
	Test Size	Time(s)	Test Size Ratio	Speedup
1	1781	9607.28	1.002	1.000
2	1787	2626.16	1.006	3.658
3	1783	2405.70	1.003	3.994
4	1777	2208.22	1.000	4.351
5	1777	2108.17	1.000	4.557

Test generation time is used to gauge the speedup gained in different machine environment. Here, the speedup is deduced from ratio of test generation time of single machine per test generation time of multi machines. These results recorded are best representative for the minimum number of test size from 10 simulations run for each setting.

From Table 1, it is evident that variations in the test size for uniform input parameter value while running on different number of machine are small as indicated by the test size ratio.The differences in term of test size is due to non-deterministic nature of the strategy resulting in different test sizes for each simulation run. There are 3 reasons that lead to this behavior. Firstly, the randomization is used to break ties in test case selection among randomly returned test cases from TS_OP Processors with same interaction coverage value in a reducer. Secondly, the randomization is also used to break ties in greedy selection among iteratively generated test case with same interaction coverage value in TS_OP Processor during horizontal or vertical extension. Thirdly, the randomization nature of interaction elements data sequence polled from space for test case generation. The random sequence of interaction elements is used to merge all possible interaction elements between each other to produce a new test case in TS_OP Processor during vertical extension.Overall, the distribution of interaction element data and test suite generation work among TS_OP Processor partition space in multiple machine environments do not degrade the optimality of the test suite solution.

Using data from the Table 1, the speedup versus number of machine is plotted as shown in Figure 5. The increment of speedup value between 2 to 5 machines is only small as compare to the increment of speedup value between 1 and 2 machines environment. This happened due to high CPU and cache memory usage while running in single machine environment. Most of the main memory is used to store the interaction element in shared memory space during test case generation. As we distributed the computing work to others machine, the main memory utilization per each machine become less. Thus permit faster computation in all available CPUs and resulted in faster time for the test case generation multiple machines.

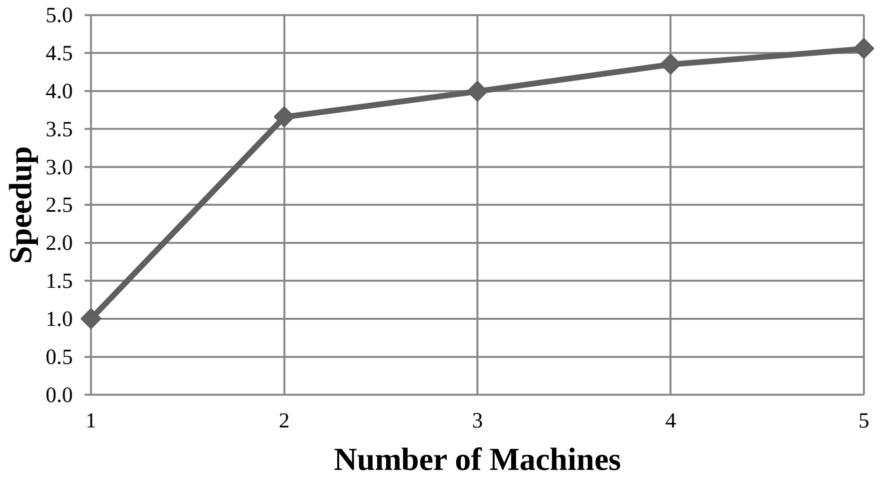

Fig. 5: The speedup for *v*=5 and *p*=10 with *t*=4 on five different machines.

4.2 Comparison with parameter based strategy

To benchmark the performance of TS_OP against other existing strategies, the TS_OP is compare with parameter based strategy such as MIPOG, IPOG, IPOG-D, IPOG-F and IPOG-F2. The main goal is to show that TS_OP *t*-way "one-parameter-at-time" strategy is sufficiently competitive as compared to other parameter based strategy in term of the generated test suite size. Furthermore, to prove that distribution of interaction elements data and the computing work among connected workstation does not compromised the optimality of test size generated in relation to other existing work. Here, a common configuration system, the TCAS module, an aircraft collision avoidance system developed by the Federal Aviation Administration is adopted as a case study similar to other related works [14]. All the test size results for TCAS value are obtained from [12] for MIPOG, IPOG, IPOG-D, IPOG-F and IPOG-F2. The best solution is bold font with less shaded area and the second best solutions in term of most optimum test suite size as shown in the darker shaded area with italic font. The experiment result for TCAS input parameter value is shown in Table 2. As seen in Table 2, for *t*=2 all strategies except IPOG-D produce a minimum test size of 100. As for TS_OP strategy, a minimum test suite size is produced for two inputs setting with interaction strength, *t*=2 and 5. As for other input setting of *t*=3, 4 and 6; the MIPOG strategy outperforms others strategies in term of most optimum test size.

Table 2: The comparison of TS_OP with other strategy for TCAS value with varying interaction strength from *t*=2 to 6.

	TS_OP	**MIPOG**	**IPOG**	**IPOG-D**	**IPOG-F**	**IPOG-F2**
t	Test Size	Test Size	Test Size	Test Size	Test Size	Test Size
2	**100**	**100**	**100**	*130*	**100**	**100**
3	408	**400**	**400**	487	*402*	427
4	1355	**1265**	1361	2522	*1352*	1644
5	**4166**	*4196*	4219	5306	4290	5018
6	11105	**10851**	*10919*	14480	11234	13310

Despite producing non-deterministic test suite even with the same input parameters, TS_OP can still produce competitive test size as compared to others strategies. By running TS_OP many times and taking a minimum test size,TS_OP strategy is able to produce the most optimum test size as shown in Table 2 for interaction strength , *t*=2 and 5.

5 Conclusion

This paper highlights a distributed strategy called TS_OP based on “one-parameter-at-a-time” for *t*-way test suite generation on single and multiple machine environments using tuple space technology. All results demonstrate that the distribution of interaction element data and computing work among participating workstation does not compromise the optimality of test suite solution (i.e. minimally affect the test suite size). A small difference in term of test suite size in the TS_OP strategy resulted from the non-deterministic nature of the strategy (i.e. the randomization nature of test case selection among test case by the reducer and the randomization in greedy selection of test case with same interaction coverage during both horizontal and vertical extension by TS_OP Processor).

The scalability analysis indicates that distribution of computing work for test suite generation across multi machine environments always give speedup as compared to single machine environment. This points out the applicability of tuple space technology as distributed shared memory platform for our application. Although the speedup does not increase linearly as compared to the number of the machine, a reasonable speedup is still obtained on multi machine environments. Comparative study between the TS_OP strategy with other strategies such as MIPOG, IPOG, IPOG-D, IPOG-F and IPOG-F2 indicates that the test suite size produced by TS_OP strategy is satisfactorily competitive in term of generating minimum test suite size.

Further works will be carried out on implementation of the strategy on cluster machine with high RAM to provide low latency whilst generating the test suite. Other than cluster machine, the strategy can also be deploy in cloud computing environment.

Acknowledgement

This research is partially funded by the generous grants – “Integrating Seamless Crash Recovery Support for T-way Test Generation Strategy” from UiTM Shah Alam, Malaysia and ERGS Grant: CSTWay: A Computational Strategy for Sequence Based T-Way Testing.

References

[1] Y. Lei, *et al.* 2007. IPOG: A General Strategy for T-Way Software Testing, *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, 2007, pp. 549-556.

[2] NIST. (2011). *Website for the NIST Automated Combintorial Testing (ACTS) project*. Available: http://csrc.nist.gov/groups/SNS/acts/index.html

[3] M. N. Borazjany, *et al.* 2012. Combinatorial Testing of ACTS: A Case Study, *Proceedings of the IEEE 5th International Conference on Software Testing, Verification and Validation (ICST)*, 2012, pp. 591-600.

[4] L. S. G. Ghandehari, *et al.*. 2013. Applying Combinatorial Testing to the Siemens Suite, *Proceedings of the IEEE 6th International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2013, pp. 362-371.

[5] Y. Lei, *et al.* 2008. IPOG/IPOG-D: Efficient Test Generation for Multi-way Combinatorial Testing, *Software Testing, Verification and Reliability*, vol. 18, pp. 125-148, 2008.

[6] M. Forbes, *et al.* 2008. Refining the In-Parameter-Order Strategy for Constructing Covering Arrays, *Journal of Research of the National Institute of Standards and Technology,* vol. 113, pp. 287-297, 2008.

[7] T. Yu-Wen and W. S. Aldiwan. 2000. Automating Test Case Generation for the New Generation Mission Software System," *Proceedings of the IEEE Aerospace Conference Proceedings*, 2000, pp. 431-437 vol.1.

[8] ITCH. 2012. *IBM ITCH*. http://www.alphaworks.ibm.com/tech/whitch.

[9] TVGII. 2012). *TVG Web Page*. http://sourceforge.net/projects/tvg.

[10] J. Czerwonka. 2006. Pairwise Testing in Real World Practical Extensions to Test Case Generators, *Proceedings of 24th Annual Pacific Northwest Software Quality Conference*, 2006, pp. 419-430.

[11] Jenny. 2010. *Jenny Web Page*. http://www.burtleburtle.net/bob/math.

[12] M. I. Younis and K. Z. Zamli. 2010. MC-MIPOG: A Parallel t-Way Test Generation Strategy for Multicore Systems, *ETRI Journal*, vol. 32, pp. 73-83, Feb 2010.

[13] GigaSpaces. 2013. *Website for GigaSpaces*.

[14] M. I. Younis and K. Z. Zamli. 2009. ITTW: T-way Minimization Strategy Based on Intersection of Tuples, *Proceedings of the IEEE Symposium on Industrial Electronics & Applications, 2009*, pp. 221-226.