International Journal of Physical Sciences Vol. 7(15), pp. 2339 - 2363, 9 April, 2012
Available online at http://www.academicjournals.org/IJPS
DOI: 10.5897/IJPS11.1316
ISSN 1992 - 1950 ©2012 Academic Journals

Full Length Research Paper

Comparative study of distributed implementation using Tuple space technology of two t-way test suite generation strategies

Zainal Hisham Che Soh[1]*, Syahrul Afzal Che Abdullah[1] and Kamal Zuhairi Zamli[2]

[1]Centre of Computer Engineering Studies, Faculty of Electrical Engineering, Universiti Teknologi MARA (Penang Campus) 13500 Permatang Pauh Pulau Pinang, Malaysia.
[2]School of Electrical and Electronic Engineering, Universiti Sains Malaysia Engineering Campus 14300 Nibong Tebal, Pulau Pinang, Malaysia.

Accepted 3 February, 2012

This paper highlights a comparative study of two new distributed t-way test suite generation strategies, called test suite generator one parameter (TS_OP) and test suite generator one test (TS_OT). Both strategies adopt computational greedy algorithm and were implemented on distributed shared memory environment using Tuple space technology. To characterize the behavior of both strategies in term of test size growth and test generation time, both TS_OP and TS_OT were subjected to a few experimentations with varied parameter, parameter value and interaction strength. Furthermore, to determine their scalability performance in term of speedup gained and test size ratio, both strategies were subjected to scalability analysis on single machine and multiple machine environments. An encouraging result on speedup is obtained for both strategies thus indicating the effectiveness of using Tuple space technology in distributing the t-way test suite generation computing work. A comparison against existing strategies in terms of the generated test suite size also indicates that both strategies give sufficiently competitive results. Furthermore, in comparison between both strategies on test suite size, TS_OP gives more satisfactory and competitive results as compared to TS_OT strategy.

Key words: T-way testing, distributed test suite generation, map and reduce, tuple space technology.

INTRODUCTION

Nowadays, software is developed in a modular way; complete working software system is achieved by integrating these small modules together. In order to construct a quality software system, a thorough integration testing with lots of test cases needs to be tested during integrating these small modules. As for highly customizable and configurable software system, such as web application (Wenhua et al., 2009; Sampath et al., 2008; Cohen et al., 2007), the number of input parameter that needs to be tested can be enormous. In both cases, large numbers of test cases are needed to test all possible software interaction in the whole system.

Indeed, a lot of testing work is required to ensure all possible software interactions between input parameters are tested to avoid faulty condition. Lacks of testing can result in an unintended or faulty interactions remain undetected and can cause the whole software system failure in future. On the other hand, exhaustive testing is next to impossible due to limited time and resources constraint (Zamli and Younis, 2010).

In order to overcome the aforementioned issues in minimizing the testing works while maintaining an adequate testing coverage, a systematic approach known as *t*-way testing can be used (Schroeder et al., 2004). The *t*-way testing (*t* indicate the interaction strength) is a systematic sampling technique to ensure fault detection of faulty interaction inside a given software system by executing its *t*-way test suite. The *t*-way test suite is a set

*Corresponding author. E-mail: zainal872@ppinang.uitm.edu.my.

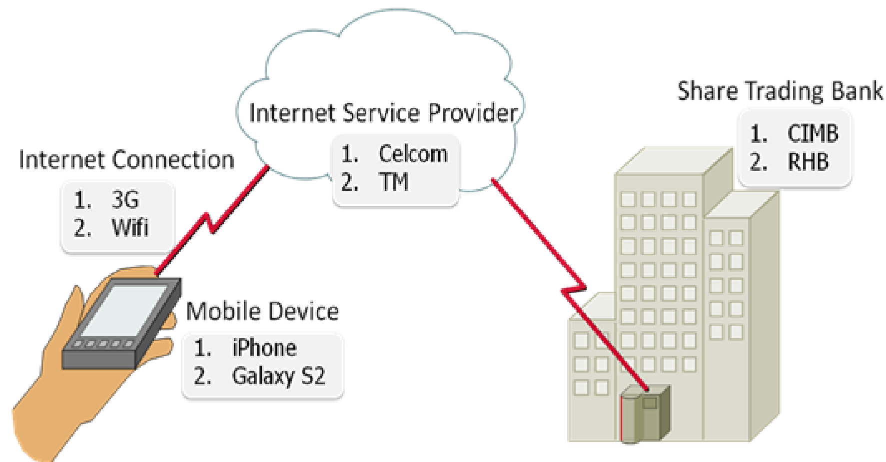

Figure 1. Mobile share trading system.

of test cases that cover all possible interaction element combination among *t* input parameters by at least one of test cases in the test suite. The pairwise or 2-way interaction testing is one form of *t*-way interaction testing that is used to test all possible interaction among two different input parameters (Czerwonka, 2006; Bryce and Colbourn, 2007; Kimoto et al., 2008; Klaib et al., 2008; Calvagna and Gargantini, 2009).

In recent time, the software size is growing fast from megabyte to terabyte due to high demand from mass public for new software application, add-on features and advance software tools. Nowadays, most of the significant software system is commonly made of millions of line code. For real, large and complex software system, the complexity and rapid software growth result in many possibility of new intertwines dependency among large software input parameter or component involved thus justifying the need to support for high interaction strength. Furthermore, there are a few works (Kuhn et al., 2004, 2009, 2008) indicate the needs for higher interaction strength to ensure full faults detection in system under test (SUT).

However, producing optimum t-way test suite for high interaction strength is a NP hard problem (Williams and Probert, 2001; Shiba et al., 2004; Kuo-Chung and Yu, 2002) and requires significant computational power and memory resources. Furthermore, as the interaction strength increases coupled with large input parameter, the possible interaction element combinations are likely to be huge and could lead toward a combinatorial explosion problem. This critical problem can possibly halt the test suite generation due to out of memory problem.

Nowadays, mostly available works on t-way testing exploits sequential algorithm, running on single or standalone machine and cannot be extend to work on multiple machine. Although useful, sequential algorithm can be counterproductive particularly when dealing with large input parameters and high interaction strength. Furthermore, a standalone machine appears to be lacking of processing power and memory space. Due to this limitation, the standalone computers were insufficient in dealing with large input parameters and high interaction strength.

In order to address these issues, we opted to develop a distributed t-way test suite generation strategy capable of distributing the computing work among participating workstations on different physical location. Here, tuple space technology is utilized to provide the parallel and distributed processing mechanism. The tuple space technology provided a generative communication mechanism between application services by mean of associative matching of tuple in a shared memory space called tuple space. The associative matching process is fast, simple and support concurrent applications.

On distributed memory system (that is, network of workstation), the tuple space data is usually distributed among the participating workstation using hashing system or partitioning system. The tuple space data can also be fully replicate to all workstation. In both scenarios, parallel or distributed processing can be achieved by assigning more applications or services to compute the data at each different workstation.

In this paper, we develop, implement, evaluate and compare two distributed t-way test suite generation strategies based on “one-test-at-a-time” strategy and “one-parameter-at-a-time” strategy for t-way testing. Both strategies described in this paper are based on map and reduce framework using tuple space technology.

FORMULATION OF T-WAY PROBLEM MODEL

A simple customizable software system is used here as a model to illustrate the idea of t-way software interaction. Figure 1 represents the topology of an online mobile share trading system.

Table 1. Share trading system components and configurations.

Input parameter	Components or parameter			
	Internet connection	Mobile device	Internet service provider	Bank
Configurations or parameter value	3G	iPhone	Celcom	CIMB
	Wifi	Galaxy S2	TM	RHB

Table 2. Test suite for t=4 for above share trading system.

Test no.	Internet connection	Mobile device	Internet service provider	Securities
1	3G	iPhone	Celcom	CIMB
2	3G	iPhone	Celcom	RHB
3	3G	iPhone	TM	CIMB
4	3G	iPhone	TM	RHB
5	3G	Galaxy S2	Celcom	CIMB
6	3G	Galaxy S2	Celcom	RHB
7	3G	Galaxy S2	TM	CIMB
8	3G	Galaxy S2	TM	RHB
9	Wifi	iPhone	Celcom	CIMB
10	Wifi	iPhone	Celcom	RHB
11	Wifi	iPhone	TM	CIMB
12	Wifi	iPhone	TM	RHB
13	Wifi	Galaxy S2	Celcom	CIMB
14	Wifi	Galaxy S2	Celcom	RHB
15	Wifi	Galaxy S2	TM	CIMB
16	Wifi	Galaxy S2	TM	RHB

The system enable share trader to buy or sell stock online via stock broking bank using a mobile device such as smart phones, tablets etc at anytime and anyplace. The system may use different components or parameters. In this paper, the term “parameter” (or P) is used to describe the components of the system. In this example, the system consists of four parameters. The mobile share trader user can use smart phones such as iPhone and Galaxy S2 as their mobile devices. For any selected mobile device, the share trading platform at CIMB or RHB securities is assessable through internet connection via 3G or Wifi provided by internet service provider such as Celcom and TM. There are different configurations in any cases. The term “value” (or *v*) is used to describe the configuration of each component. Thus, the system in Figure 1 can be summarized as a four-parameter system with two values as in Table 1, to illustrate how *t*-way testing works, and hence demonstrate the test case reduction, for the aforementioned example. Here, the range of acceptable *t* is between two to maximum parameter number which is four. Tables 2 to 4 shows the examples of the generated test cases for *t*-way testing with varying t value of four, three and two respectively. *t*=4, here can also be refer as exhaustive testing, it involves explicit enumeration of all possible combinations of their input parameter and for this case the number of test case is 16 test cases (that is,, 2 × 2 × 2 × 2). Using t-way testing approach, for *t*=3, their test case number is reduce to only 8 test cases and adhere to 3-way interaction coverage. As for t=2, as normally called pairwise testing, their test case is further reduce to only 6 test cases. As a result, it is added advantages to utilized t-way testing to reduce the test case number while maintaining adequate interaction coverage.

RELATED WORKS

As Lei et al. (2007) briefed in his paper, there have been two strategies for generation of test suites either computational or algebraic approach. Both computational and algebraic approaches have their own advantages and disadvantages such as computational approaches can be applied to any input system configurations, but the computation can be intensive. Algebraic approaches on the other hand usually involved lightweight computations and in some cases, algebraic approaches can produce optimal test sets. However, algebraic approaches often impose restrictions on input system configurations to which they can be applied.

In algebraic approaches, test suite is constructed using

Table 3. Test suite for t=3 for above share trading system.

Test no.	Internet connection	Mobile device	Internet service provider	Securities
1	3G	iPhone	Celcom	CIMB
2	3G	iPhone	TM	RHB
3	3G	Galaxy S2	Celcom	RHB
4	3G	Galaxy S2	TM	CIMB
5	Wifi	iPhone	Celcom	RHB
6	Wifi	iPhone	TM	CIMB
7	Wifi	Galaxy S2	Celcom	CIMB
8	Wifi	Galaxy S2	TM	RHB

Table 4. Test suite for t=2 for above Share Trading System.

Test no.	Internet connection	Mobile device	Internet service provider	Securities
1	3G	iPhone	Celcom	CIMB
2	3G	Galaxy S2	TM	RHB
3	Wifi	iPhone	TM	CIMB
4	Wifi	Galaxy S2	Celcom	RHB
5	Wifi	Galaxy S2	Celcom	CIMB
6	3G	iPhone	TM	RHB

pre-defined rules using mathematical function such as Latin square (Mandl, 1985), orthogonal array (Burroughs et al., 1994) and graph theory (Meagher and Stevens, 2005) to produce a t-way test suite. Other algebraic approaches are based on the idea of recursive construction based on orthogonal arrays, which allows larger test sets to be constructed from smaller ones such as TConfig (Williams, 2000; Aguirre et al., 2009).

Unlike algebraic approaches, computational approaches often rely on generating all possible interaction elements and search the entire interaction element combinations to generate the test suite until all interaction elements are covered. There are a few search techniques that can be utilized such as greedy algorithm or artificial intelligence technique. Artificial intelligent technique usually start from a pre-existing test suite and then apply a series of transformations using a fitness function to determine the test suite until a complete test suites is reached that covers all the combinations. Strategies that adopted artificial tracking techniques such as GAPTS (McCaffrey, 2009), Tabu Search (Nurmela, 2004), Ant Colony Algorithm (ACA) (Shiba et al., 2004), Genetic Algorithm (GA) (Shiba et al., 2004), Simulated Annealing (SA) (Cohen et al., 2003), and augmented annealing (Cohen et al., 2008) are proposed in the literatures. Briefly, these strategies start from some known test set. Then, a series of transformations were applied (starting from the known test set) until an optimum set is reached to cover all the interaction elements. Unlike AETG and IPOG, which build a test set from scratch, artificial intelligence search strategies can predict the known test set in advance. As such, these search techniques can produce smaller test sets than AETG and IPOG, but they typically take longer time to complete. In addition, they can only support small parameters and values, with low interaction strength. SA, Tabu Search, ACA and GA reported result with interaction strength up to 3-way coverage only.

For greedy algorithm, there are two categories of greedy algorithm for test suite generation known as "one-test-at-a-time" strategy which build test suite one test case at a time until all interaction elements are covered and "one-parameter-at-a-time" strategy which extend the test case by one parameter at a time until all parameter and interaction elements are covered. Typical "one-test-at-a-time" is exemplified by Automatic Efficient Test Generator (AETG) which iteratively builds a complete test case using greedy search technique until all the interaction element combinations are covered (Cohen et al., 1997, 1996), TCG (Yu-Wen and Aldiwan, 2000), DDA (Bryce and Colbourn, 2007). Because AETG uses random search, the generated test case is highly non deterministic.

Contradictorily, both TCG and DDA produce a deterministic test suite results due to fixed rule in generating a test case that cover as many as possible uncovered interaction elements in their greedy search of maximum interaction coverage. Bryce and Colbourn (2009) develop

an enhance DDA with support for higher interaction strength for t-way testing(. Due to random insertion of first value into the test case, higher strength DDA was unable to produce a deterministic test suite results as its predecessor DDA. Zamli et al. (2011) developed the GTWay by merging the interaction element based on their interaction element group to construct the test case with aims of higher interaction coverage. Furthermore, GTWay also provide an execution support for automatic execution of the generated test suite.

Czerwonka developed a freeware tool named PICT whose core algorithm is based on greedy algorithm and similar to AETG with key differences that PICT is a deterministic and does not produce any candidate test. PICT had rich features such as support variable strength generation, support constraint and seeding (Czerwonka, 2006). Hartman and Raskin (2004) developed the combinatorial test services (CTS) package that construct a t-way test suite using direct and recursive construction algorithm. In solving the t-way test suite construction, the CTS package tries several alternatives and chooses the smallest array that is constructed. All t-way test suite with similar input configuration always produce similar size for each new construction due to all the algorithms employed are deterministic. An extension of CTS known as the IBM's intelligent test case handler (ITCH) is available in Eclipse Plug-in tool (ITCH, 2010). ITCH uses a sophisticated combinatorial algorithm based on combination of mathematical and greedy search to construct the test suites for t-way testing. Other tool within these category with limited literature work but can be downloaded at their respective web site are Test Vector Generator (TVGII) (TVGII, 2010) and Jenny (2010). Both Jenny and GTWay is developed using C language.

Other group of greedy algorithm is categorizes as "one-parameter-at-a-time" strategy. These are exemplified by IPOG (Lei et al., 2007). The IPOG strategy is generalized from IPO (Lei and Tai, 1998). In this strategy, a t-way test suite for the first t parameters is generated, and then in horizontal extension phase, each test case is added with a new parameter value at t+1 parameter that covers maximum uncovered interaction elements. The newly extended test case is selected and stored into new t-way test suite. If after all test cases are extended and there are still uncovered interaction elements, then the vertical extension phase 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. If next parameter t+2, exists, then the horizontal phase and vertical phase is resume for that parameter. The test suite generation goes on until all parameter are covered.

A number of variants have also been developed to improve the IPOG's performance. These variants includes, IPOG-D (Lei et al., 2008), IPOG-F (Forbes et al., 2008), IPOG-F2 (Forbes, 2008), MIPOG, G_MIPOG (Younis et al., 2008) and MC-MIPOG (Younis and Zamli, 2010). IPOG-D is a deterministic strategy that combines the IPOG strategy with a recursive D-construction to minimize the number of interaction element that need to be enumerated during the generation of the test suites. The D-construction approach is a recursive procedure that can be used to double the number of parameters in a 3-way test suite. Although IPOG-D can generate the test suite faster than IPOG, their test size is usually bigger than IPOG.

Both IPOG-F and IPOG-F2 are non-deterministic strategies. Both strategies implemented a randomization to break ties in the greedy selection during the horizontal growth. In general, the size of test suite generated by both strategies is competitive as compared to IPOG. As for their execution time, they seem to be faster than IPOG. Although both strategies can support uniform and mixed input parameter setting, the performance gain 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 permitting faster test generation time as compare to IPOG-F.

MIPOG strategy is a deterministic strategy that implied that each run will produce the 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, MIPOG strategy optimization does not cover the value by searching for uncovered interaction element that can be covered by the same test case. This is performed by means of exhaustive searching of uncovered interaction element that can be combined with this test case during horizontal extension. In vertical extension, MIPOG created a new test case by exhaustively search 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 execution time of MIPOG.

Both G_MIPOG and MC-MIPOG are built based on MIPOG strategy and can parallelize the test suite generation work. G_MIPOG is implemented on a grid network while MC-MIPOG is run on an Intel 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. Other related work and the current state of combinatorial testing are given in Nie and Leung (2011) and Grindal et al. (2005).

THE OVERALL STRATEGY OF BOTH TS_OP AND TS_OT

Here, we describe about two distributed strategies for generating t-way test suite in t-way testing called Test Suite Generator One Parameter (TS_OP) for "one-

parameter-at-a-time" strategy and called as Test Suite Generator One Test (TS_OT) for "one-test-at-a-time" strategy. In both strategies, the distributed processing is implemented using Map and Reduce mechanism running on network of workstations using Tuple Space technology middleware known as GigaSpaces XAP 8.0. (GigaSpaces, 2010). Both overall strategies, TS_OP and TS_OT will be elaborated further in the subsequently.

ONE PARAMETER AT A TIME, TS_OP STRATEGY

The examples and flowchart of the overall design approach of TS_OP strategy is illustrated in Figure 2. In this strategy, a master process is called TS_OP Feeder and the worker process is called TS_OP Processor.

Initially in our overall design approach of TS_OP strategy, the TS_OP Feeder preloads the assigned parameter value, v_i, the interaction strength, t and the input parameter data, *ParmSet* into all TS_OP Processor dedicated partition space. For the illustration examples in Figure 2 that use a small input parameter with 4 parameters (that is, P_0 P_1 P_2 P_3) and 2 parameter values (that is, v_0 v_1). The interaction strength, t used is 3.

In horizontal extension, The TS_OP Feeder generates t-way test suite for the first t-parameters and store them in *TS*. In illustration, the test suite is generated from the first three parameters (that is, P_0 P_1 P_2) and produces eight test cases as shown in initial *TS* at master space. Each test case in *TS* is individually sends to all TS_OP Processor one by one.

The TS_OP Feeder remotely executes all worker processes by sending a command *generateIE* with current parameter, P_i to all TS_OP Processors concurrently. Each TS_OP Processor generates all t-way interaction element (ie) combinations between P_i and using the assigned parameter value, v_i and stores them in their respective partition space as interaction element set, *ieSet*. As shown in initial population phase, the *ieSet* in each worker space is created with list of 3-way interaction element combination. The current parameter, P_i is the P_3 parameter with two parameter value. All interaction element combination must contain parameter P_3 value and two other parameters value as shown in both *ieSet*.

After initial population phase, the generation of test case phase follows. Here, the TS_OP Feeder sends a command *constructTestCase* that consists one of generated test case, τ and the current parameter, P_i to all TS_OP Processor concomitantly to extend test case. For test case without don't care value, each partition space constructs the new test case by adding the assigned parameter value to test case, τ. Next, the maximum interaction coverage value is calculated for newly extended test case.

The calculation of interaction coverage is done by matching the interaction element covered by generated test case against interaction element in *ieSet* at their partition space. In order to obtain all the interaction element covered by that test case, the test case is factorize into individual interaction element data based on their interaction element group. All individual interaction element are match against the interaction element in their respective partition space, the maximum interaction coverage, *maxIEC* is determined by the number of matched interaction element between interaction element of factorise test case with interaction element in their partition space. In the illustration in Figure 2, the *maxIEC* for $T1_0$ [0 0 0 0] is 3 and the *maxIEC* for generated test case at v_1 worker space for $T1_1$ [0 0 0 1] is also 3.

As for test case with don't care value, the test case, τ will be optimizes into τo by merging it with possible interaction element in *ieSet* at their respective partitions. Then the maximum interaction coverage is calculated as aforementioned.

All space remoting results contain the maximum interaction coverage, *maxIEC* and the selected test case, τ' from their respective partition are return to calling TS_OP Feeder via a Reducer. The Reducer selects one test case with highest interaction coverage value among results returned by all available TS_OP Processors.

As for our illustration in Figure 2, the *maxIEC* for both test cases are three. Therefore, any test case will be randomly selected into final *TS*. Here test case $T1_0$ is selected into final *TS*.

The TS_OP Feeder stores the selected test case, τ' or τo into the final test suite, *TS* and deleted all interaction element combinations covered by that test case in all partition space. Then the TS_OP Feeder iteratively sends all test cases in *TS* to all TS_OP Processors and completes the horizontal extension phase.

After all test cases in TS already delegated to all TS_OP Processors, the horizontal extension phase is completed, if there are still uncovered interaction element combinations in the *ieSet*, then the vertical extension phase will commence.

As shown in the illustration in Figure 2, the first test case T1 [0 0 0] is sent to both workers space. For test case T1, there is no don't care value in the test case which result in only insertion of assigned parameter value at P_3 for both worker space. The generated test case at v_0 worker space is $T1_0$ [0 0 0 0] and the generated test case at v_1 worker space is $T1_1$ [0 0 0 1].

From here, both worker spaces continue to calculate the maximum interaction coverage covered by their generated test case. Both test cases have same maximum interaction coverage of three. Here, reducer randomly selected one test case and stored in *TS*. The selected test case is $T1_0$ [0 0 0 0]. As a result of this test case selection, the three covered interaction element, (that is, [0 0 X 0] [0 X 0 0] [X 0 0 0]) in ieSet of v_0 worker space are deleted.

The TS_OP Feeder checks the availability of interaction element in *ieSet* at all partition space. If not available, the test case generation is stopped, however

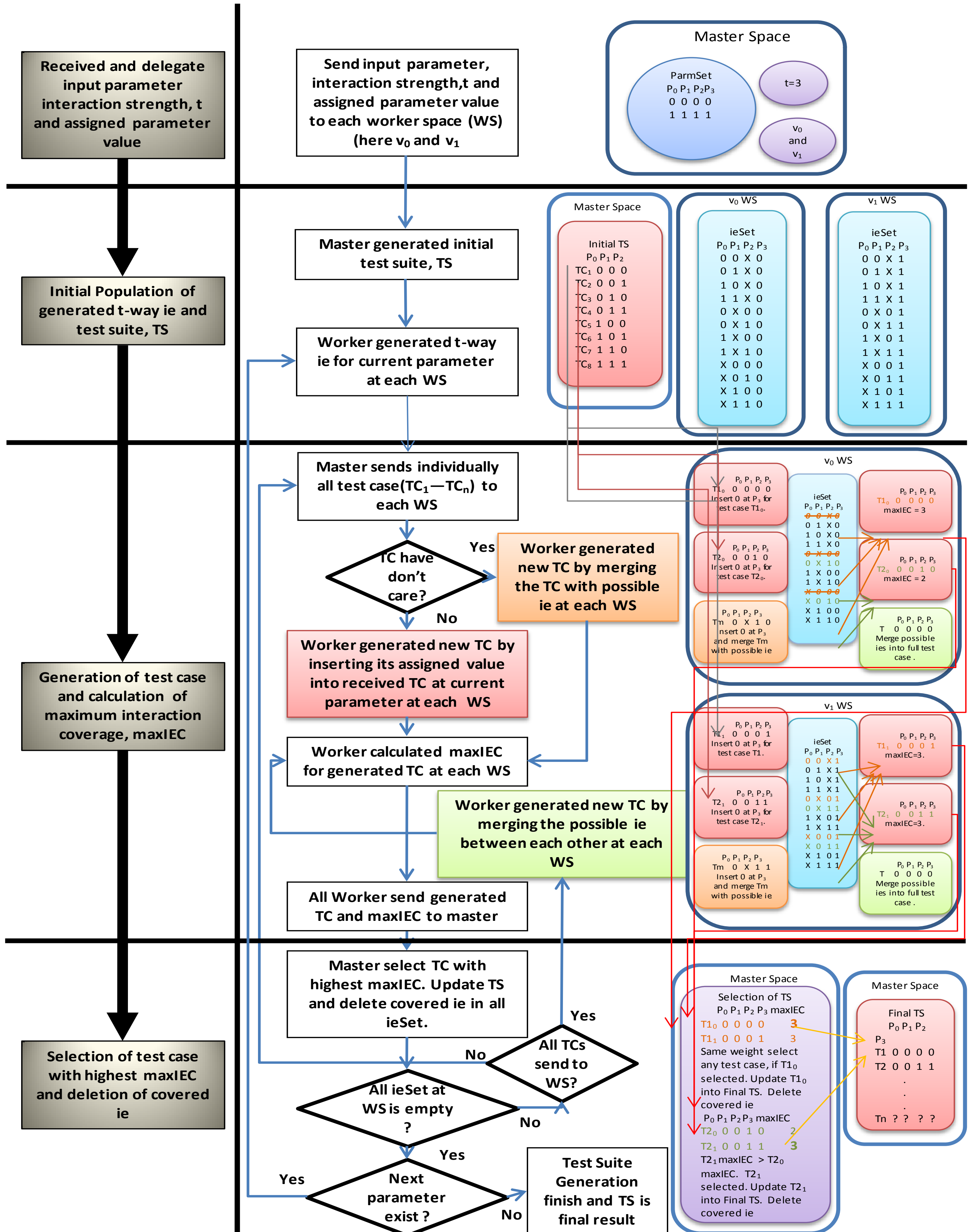

Figure 2. An illustration example and flow chart of TS_OP strategy.

if there are interaction element in *ieSet*, the TS_OP Feeder further checks for the availability of unsent and not extended test case in TS, if available the test case generation is continued and if not available the vertical extension is started.

In vertical extension phase, The TS_OP Feeder sends a command *calculateIECMax* with the *maxIEC* value equal to maximum number of interaction element group to all TS_OP Processor concurrently to merge possible interaction element data stored in their respective partition space into one complete test case and to calculate the maximum interaction coverage from generated test case. All results contain the maximum interaction coverage; *maxIEC* and the selected test case, *tsm* from their respective partition are returned to calling TS_OP Feeder via a Reducer. The Reducer selected one test case with the highest interaction coverage value among results returned by all available TS_OP Processor. The TS_OP Feeder stores the test case, *tsm* into temporary test suite, *TS'* and deletes all interaction element combinations covered by that test case in all partition space in order to ensure optimal solution.

The TS_OP Feeder then continues to remotely execute all TS_OP Processor to obtain next *tsm* until all interaction element combinations are covered. In order to minimise the search work for the next *tsm*, the previous maximum interaction coverage value, *maxIEC* is sends to all TS_OP Processor. The previous attainable *maxIEC* value is uses as the stopping criteria during generation of the *tsm*. If the current *maxIEC* is equal or greater than previous *maxIEC*, the greedy search of the interaction element combinations is halt and the generated test case is returns to TS_OP Feeder along with the current value of *maxIEC* value. Otherwise, the greedy search is continued until all interaction elements are tested and the latest test case is selected and returns to the TS_OP Feeder.

After all interaction elements are covered and *ieSet* is empty in all partition spaces, the extension of the current parameter, P*i* is completed then the test case generation work will continues to next parameter, P*i*+1. The resultant temporary test suite, *TS'* is also minimises by removing redundant *tsm*. The minimized *TS'* is then added into the final test suite, *TS*. This strategy will continue until the last parameter, P*t* is accounted for and the complete test case is formed. Finally, the TS_OP Feeder displays the final test suite, TS and stores the test suite into testsuite log file.

The complete algorithm for TS_OP is given as follows:

Algorithm *TS_OP(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. TS_OP Feeder send command generate IEs to all TS_OP Processors
8. TS_OP Processor receive command generate IEs and *Pi* from TS_OP Feeder through space based remoting*;*
9. generate t-way ie between Pi and{P1…Pt} using assigned parameter value and stored into in *ieSet* TS_OP Processors partition;
10. \\ let *ieSet* be the set of all pair combinations of values between *Pi* and each of *P1, P2,…,Pi-1*;
11. **for** each test *τ = (v1, v2, …, vi-1)* in test suite *TS* **do**
12. TS_OP Feeder send command construct test case and *τ* to all TS_OP Processor *;*
13. TS_OP Processor receive command construct test case and *τ* from TS_OP Feeder through space based remoting*;*
14. **if** (*τ* not contains don't care)
15. insert assigned value (*v*) into *τ;*
16. calculate the *maxIE*;
17. send *τ'* and *maxIE* to TS_OP Feeder via reducer*;*
18. **else** merge *τ* with possible interaction element combination in *ieSet* into *τo* ;
19. calculate the *maxIE*;
20. send *τo* and *maxIE* to TS_OP Feeder via reducer*;*
21. reducer choose the *τ'* with highest *maxIE;*
22. add selected test case *τ'* to *TS;*
23. delete all covered interaction element in *ieSet* ;
{ *vertical extension for parameter Pi* }
24. **while** *(ieSet is* not empty*)* **do**
25. TS_OP Feeder send command calculate IEC Max to all TS_OP Processor;
26. TS_OP Processor receive command calculate IEC Max from TS_OP Feeder through space based remoting
27. merge possible interaction element combination in *ieSet* into *tsm* ;
28. calculate the *maxIE*;
29. send *tsm* and *maxIE* to TS_OP Feeder via reducer*;*
30. reducer choose the *tsm* with highest *maxIE;*
31. add selected test case *tsm* to *TS';*
32. delete all covered interaction element in *ieSet* ;
33. remove redundant tsm in *TS'*
34. add temporary *TS'* to *TS*
end
35. **return** *TS;*
end

ONE TEST AT A TIME, TS_OT STRATEGY

Here, the design approach of the test suite generation strategy based on "one-test-at-a-time" strategy called Test Suite Generation One Test (TS_OT) strategy is been explained. The overall strategy and example of TS_OT

strategy is illustrated in Figure 3. The strategy is also implemented using Map and Reduce mechanism on network of workstations using Tuple Space technology middleware known as GigaSpaces XAP 7.0 (GigaSpaces, 2010). In this strategy, a master process is called TS_OT Feeder and the worker process is called TS_OT Processor.

In overall design approach, the TS_OT Feeder preloads the interaction strength, *t* value and the input parameter data, *ParmSet* in POJO format into all TS_OT Processor dedicated partition space. The TS_OT Feeder generates all possible interaction element combinations and delegate the interaction element into TS_OT Processor partition space based on hash table based routing mechanism. As shown in our illustration in Figure 3, for small input parameter with 4 parameters (that is, P_0 P_1 P_2 P_3) and 2 parameter values (that is, v_0 v_1) with the interaction strength, *t* equal to 3. Here, the TS_OT Feeder generates all 32 interaction element combinations. For two partition space as in our examples, each partition will receive 16 interaction elements from TS_OP Feeder with worker 1 space obtained all interaction elements with odd number and worker 2 space obtained all interaction elements with even number.

The TS_OT Feeder sends a command *calculateIECMax* to all TS_OT Processor concurrently to generate the test case by merging possible interaction element data within their respective space partition. Each TS_OT Processor merged the interaction element according to their interaction element group. The first interaction element is taken from list of uncovered interaction element fetch from their partition space. The first interaction element is then merged with others interaction elements according to the interaction element group sequence to generate the new test case. If the new test case contained no don't care 'X' then the test case generation is exited otherwise the test case generation will continue and stopped when all the interaction element group in the list are all tried and tested for merging possibility with currently built test case. In our illustration and example in Figure 3, the first interaction element poll from *ieSet1* in worker 1 space is IE1 [0 0 0 X] and try to be merged with the next interaction element in the list which is [0 1 0 X] but unsuccessful and next interaction element [1 0 0 X] and [1 1 0 X] but still unsuccessful until interaction element [0 0 X 0] and produce one complete test case of [0 0 0 0]. For worker 2 space, test case produce is [0 1 1 0].

Next, the maximum interaction coverage value is calculated for newly generated test case in each TS_OT Processor space partition. The calculation of interaction coverage is done by matching the interaction element covered against interaction element in *ieSet* at their partition space. In order to obtain all the interaction element covered by that test case, the test case is factorize into individual interaction element data based on their interaction element group. All individual interaction element are match against the interaction element in their respective partition space; the maximum interaction coverage, *maxIEC* is determined by the number of matched interaction element of factorize test case with interaction element in their partition space. In our example, the maximum interaction coverage value for worker 1 space is 4 and worker 2 space is 4.

The current determined *maxIEC* value is then compared with value of the previous *maxIEC* value. If the current *maxIEC* is equal or greater than previous *maxIEC*, then the current *maxIEC* value and its test case are stored as the best solution so far. Otherwise the previous *maxIEC* value and its test case are stored as best solution.

From here, the selected *maxIEC* value is then compared with the best maximum interaction coverage from previous selected test case in final test suite, *TS* labeled as previous attainable maximum interaction coverage, *pamaxIEC.*

The value of previous attainable maximum interaction coverage, *pmaxIEC* value is used as the stopping criteria of current test case generation. If the current *maxIEC* is equal or greater than *pmaxIEC*, then the current *maxIEC* value and its test case are considered as the best solution and test case generation is stopped. Here the value of *pmaxIEC* for our example is 4, so both generated test case is sent to reducer. The generated test case returns to TS_OT Feeder along with the current value of *maxIEC* value via a reducer. Otherwise, the test case generation is continued 15 times and the highest value of *maxIEC* among the 15 solution and its test case is selected and returns to the TS_OT Feeder.

All space remoting results containing the maximum interaction coverage; *maxIEC* and the selected test case, *τ* from their respective partition are return to calling TS_OT Feeder via a Reducer. The Reducer selects one test case with highest interaction coverage value among results returned by all available TS_OT Processor. For cases with same *maxIEC,* the test case is randomly selected. For examples as illustration in Figure 3, the test case [0 0 0 0] with *maxIEC* of 4 is randomly selected and stored in *TS*. The TS_OT Feeder stores the selected result containing one test case, *τ* and its maximum interaction coverage value, *maxIEC* into the final test suite, *TS* and deleted all interaction element combinations covered by that test case in all partition space. The TS_OT Feeder then continues to remotely execute all workers to obtain *τ* and its *maxIEC* at their respective partition until all interaction element combinations are covered. The generation of test suite will stop when the interaction element data, *IESet* in all space partition is empty. Finally, the TS_OT Feeder displayed the final test suite, TS and stored in test result log file

The complete algorithm for TS_OT is given as follows:

Algorithm *TS_OT Feeder/Master (ParameterSet ParmSet, t)*

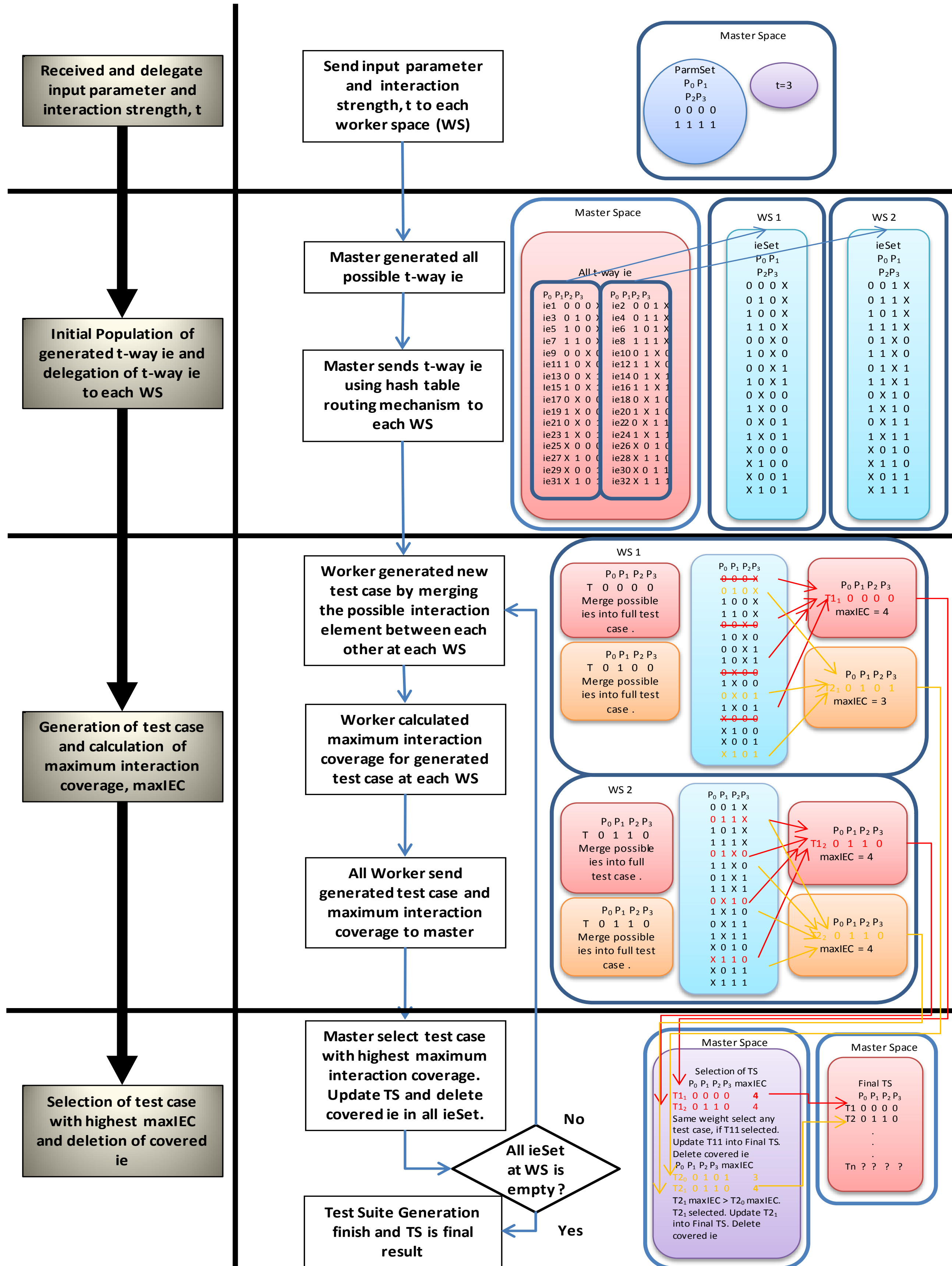

Figure 3. An illustration example and flow chart of TS_OT strategy.

```
begin
1.   initialize test suite TS to be an empty set;
2.   denote the parameters in ParmSet, in an arbitrary order,
      as P1, P2, …, and Pn;
3.   send ParmSet and t to all partition space;
4.   generate ie and delegate by id to respective space
      partition;
5.   while (IESet is not empty) do
     begin
6.        send command calculate IEC Max to all TS_OT
          Processor;
7.        receive command calculate IEC Max from TS_OT
          Feeder through space based remoting;
8.        for (iteration lj , j=j+1, …,15) do
9.          begin
10.           merge possible interaction element combination
              in IESet into complete test case, τ;
11.           calculate the current maxIEC highest interaction
              coverage;
12.           compare current maxIEC with previous maxIEC;
13.             if (current maxIEC is higher than or equal to
                previous maxIEC)
14.                 stored current maxIEC and its test case , τ;
15.           else stored previous maxIEC and its test case , τ;
16.           compare current maxIEC with pmaxIEC;
17.             if ( selected current maxIEC is higher than or
                equal to previous pmaxIEC
18.                 send current maxIEC and its test case , τ
                    to TS_OT Feeder PU via a reducer
19.             else continue test case generation
20.         end
21.      send current highest maxIEC and its test case, τ to
         TS_OT Feeder PU via a reducer;
22.           add τ test case into TS;
23.           delete covered interaction element in all partition
              of their respective IESet;
     end
24.  Compact or deleted redundant test case within TS;
25.  display TS;
End
```

DISTRIBUTED TS_OP AND TS_OT STRATEGY

Based on the aforementioned overall strategies, an application model for each TS_OP and TSOT strategy on single machine environment is designed. The TS_OP comprises of TS_OP Feeder PU and TS_OP Processor PU with collocated partition space as shown in Figure 4. The TS_OT strategy is consist of TS_OT Feeder PU and TS_OT Processor PU as shown in Figure 5. All the Processing Unit (PU) are wired together using a Spring configuration file, *pu.xml*. The development and initial simulation of the test generation work is done in Eclipse IDE before been packaged and deployed into Gigaspaces Service Grid.

The TS_OP Feeder PU is responsible to feed all data into shared space using the *InputData Loader* services and controls test suite generation using *TestData 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 *IEC Processor* services.

As for TS_OT strategy, the TS_OT Feeder PU is designed to feed the data into shared space and to controls test suite generation using *TestData Feeder* services whereas the TS_OT Processor PU is designed to generate the test case by using *IMIECProcessor* services. All the services are loosely connected to all the data in space such as *t*, *vi*, *ParmSet* and *ieSet*.

Using the aforementioned application model as our basis, distributed TS_OP and TSOT strategies are been designed and implemented. The first step in distributing both strategies is to identify the number of worker process to represent the number of partition space needed during test suite generation. In TS_OP strategy, the number of worker process or TS_OP Processor PU is deduced from the highest number of parameter value among given input parameter. The highest number of worker process also represents the number of partition space needed in test suite generation. In TS_OT strategy, the number of worker process can be selected from 1 worker to any number of workers as needed. The maximum number of worker process or TS_OT Processor PU is only limited to the number of available and connected physical machine.

Here, both distributed TS_OP and TS_OT strategy are been implemented using a partitioned topology. In partitioned topology, the main TS_OP or TS_OT Processor PU and its partition space is divided into several block of Processing Unit with their dedicated partition space. The partitioned topology enables the storage of large volume of data by splitting the data across several TS_OP Processor PU partition spaces on different physical machine. Therefore, the large storage space can prevent the combinatorial explosion problem during test suite generation.

The second step is to identify the all distributed and common data that reside at each partition space. The distributed data in TS_OP strategy is each individual parameter value from selected parameter with highest number of parameter value. Each partition space is assigned with one uniquely assigned input parameter value, *vi* of the highest number of parameter value. This unique value *vi* is used to generate the corresponding *ieSet* for that partition space. Hence, the interaction elements exist in each partition different compared to others partition space. The distributed data in TS_OT strategy is the interaction elements with their identification (id) number. The interaction elements are been routed among each partition space according to their unique interaction element and identification number using hash table routing mechanism. In both strategies, 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 both strategies is to identify all tasks running on both TS_OP or TS_OT Feeder PU

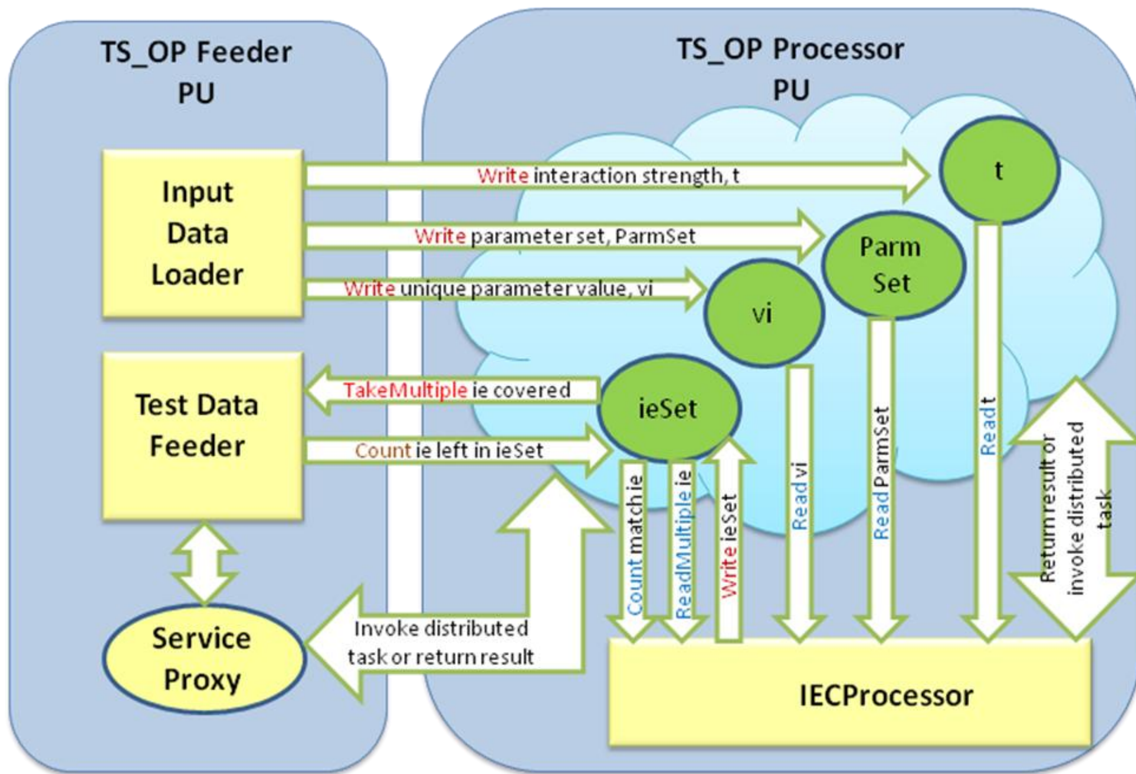

Figure 4. An application model of one TS_OP Feeder PU and one TS_OP processor PU.

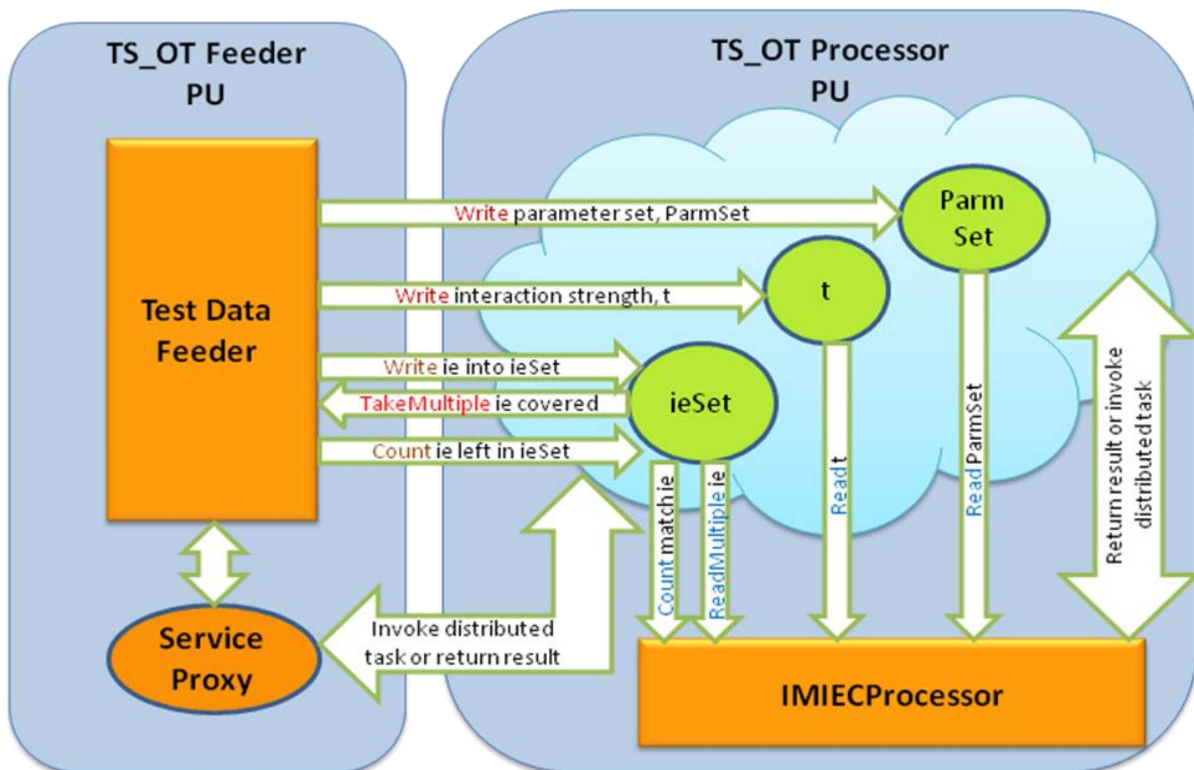

Figure 5. An application model of one TS_OT Feeder PU and one TS_OT processor PU.

and TS_OP or TS_OT Processor PU. All task services running on the TS_OP or TS_OT Feeder PU are considered as a master task whereas all tasks running on TS_OP or TS_OT Processor PU are considered as a distributed task. The similar master tasks that runs on both TS_OP and TS_OT Feeder PU are:

1. Preload of the interaction strength, *t*, the input parameter; *ParmSet* to all partition space.
2. Control of test generation and remotely execution of all distributed task on TS_OP or TS_OT Processor PU.
3. Selection and storage of selected test case in final test suite, *TS*.
4. Deletion of the interaction element covered by selected test case in all partition space.
5. Removal of redundant test case in final test suite.
6. Display and storage of the final test suite into log file.

The master tasks services that are only running on TS_OP Feeder PU are delegation of the unique value, vi to all partition space, generation of the t-way test suite for the first t parameter and removal of redundant test case after vertical extension. The master tasks services that are uniquely running on the TS_OT Feeder PU are generation of all possible interaction elements and delegation of all generated interaction elements based its id into all partition spaces using hash table routing mechanism.

The distributed task services are the task that is running on TS_OP or TS_OT Processor respective partition space and been initiated and executed by TS_OP or TS_OT Feeder PU. The synchronous mode of space based remoting service known as Map and Reduce mechanism is utilized by TS_OP and TS_OT Feeder PU to simultaneously invoke the distributed task service on all their Processor PU. A distributed processing is achieved while running in this mode.Four tasks are running on TS_OP Processor PU as distributed tasks such as:

1. Generation of the interaction element data using assigned unique value, *vi*.
2. Generation of the extended test case by inserting the assigned unique value, *vi* to that test case, calculation of the maximum interaction coverage value for the test case generated and return selected test case with maximum interaction coverage to TS_OP Feeder PU via a Reducer.
3. Generation of the extended test case with don't care by merging with possible interaction element combinations, calculation of the maximum interaction coverage value for the test case generated and return selected test case with maximum interaction coverage to TS_OP Feeder PU via a Reducer.
4. Generation of complete test case by merging between possible interaction element combinations, calculation of the maximum interaction coverage value for generated test case, selection and return of test case with highest value of maximum interaction coverage to TS_OP Feeder PU via a Reducer.

The task services designed to work on TS_OT Processor as distributed tasks are the generation of the complete test case by merging between possible interaction element combinations, 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_OT Feeder PU via a Reducer.

Finally, the distributed TS_OP and TS_OT strategies are implemented and deployed on a partitioned topology using the Giga Spaces Service Grid. The Service Grid is a set of Grid Service Container (GSC) that is managed by a Grid Service Manager (GSM). For a single machine environment, the TS_OT Feeder PU and the TS_OT Processor PU with its collocated partition space run on GSC within the same machine. For multiple machine environments, TS_OT Feeder PU and each TS_OT.

Processor PU with their respective partition spaces are distributed across several different physical machines in different GSC. As an illustration for multiple machine environments of a distributed TS_OP strategy with three workers is shown in Figure 6. The TS_OP Feeder PU has two collocated services, *InputData Loader* services and *TestData Feeder* services in GSC 4 on Machine 4. The complete algorithm for all services running in TS_OP Feeder PU is given in Figure 7. As for all three TS_OP Processors PU with their collocated partition spaces running on Machines 1 to 3 respectively, the complete algorithm for the *IECProcessor* service is shown in Figure 8.

Illustration for TS_OT strategy with multiple machine environments is shown in Figure 9. The TS_OT Feeder PU has one service, *TestData Feeder* service in GSC 4 on Machine 4. The complete algorithm for the services running in TS_OT Feeder PU is explained in Figure 10. As for all three TS_OT Processors PU with their collocated partition spaces running on Machines 1 to 3 respectively, the complete algorithm for the *IMIECProcessor* service is shown in Figure 11.

EVALUATION

Here, three group of experiments have been carried out to access the performance and behaviour of both developed strategies. The first group of the experiment is done to characterize both strategies performance in term of their test size growth and their test generation time. The second group of experiments were done to access the scalability of both strategies in term of speedup gained while running on a multiple machine environments. The third group of experiments were carried out to compare both developed strategies with existing strategies in term of size of generated test suite.

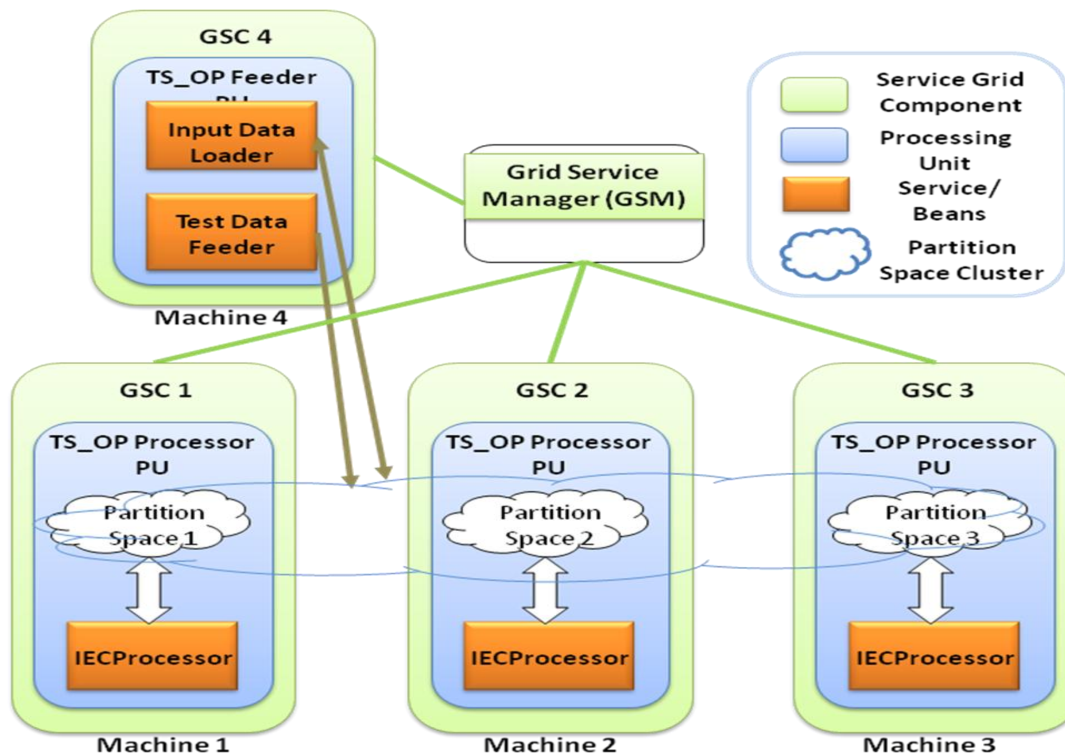

Figure 6. The TS_OP strategy implementation for multiple machine environments.

Algorithm *TS_OP Feeder (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 Processors 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

Figure 7. An algorithm of TS_OP feeder/master.

```
Algorithm  TS_OP Processors
begin
   1. initialize as one 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 =t+1,…,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
```

Figure 8. An algorithm of TS_OP processor/worker.

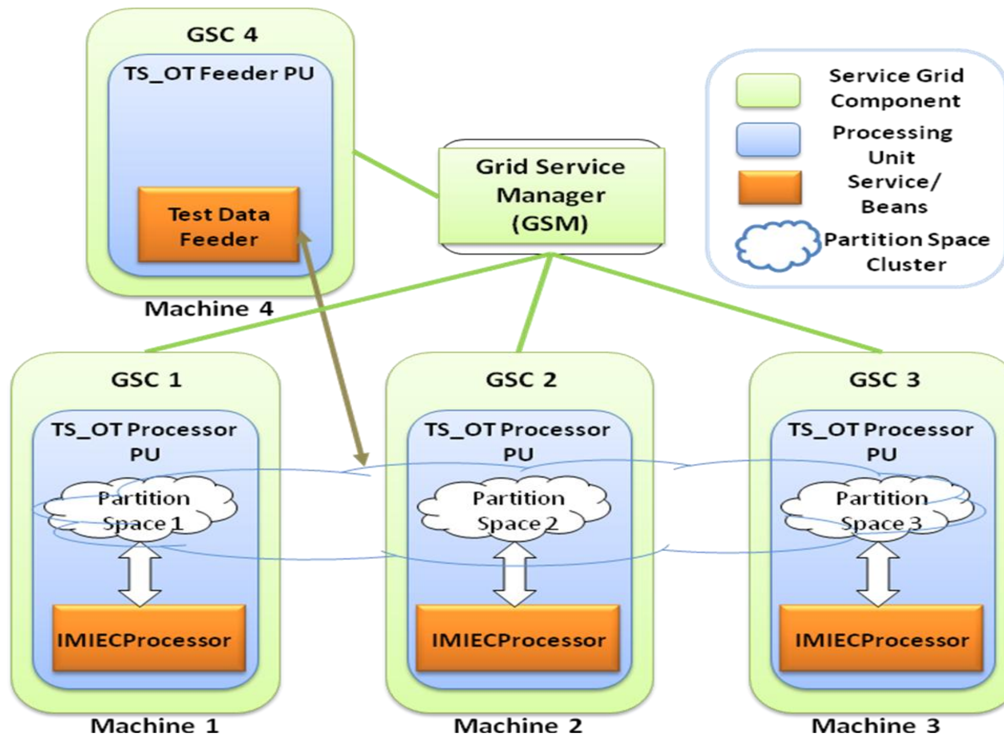

Figure 9. The TS_OT strategy implementation for multiple machine environments.

```
Algorithm TS_OT Feeder/Master ( ParameterSet ParmSet, t)
begin
1.      initialize test suite TS to be an empty set;
2.      denote the parameters in ParmSet, in an arbitrary order, as P1, P2, …, and Pn;
3.      send ParmSet and t to all partition space;
4.      generate ie and delegate by id to respective space partition;
5.      while (IESet is not empty) do
    begin
6.      execute a worker processor at their respective partition space to generate a test case
by merging possible interaction element find to test case, τ with highest interaction coverage;
7.      wait for result of τ with maxIEC from a reducer;
8.      add τ test case into TS;
9.      delete covered interaction element in all partition of their respective IESet;
        end
10.     Compact or deleted redundant test case within TS;
11.     display TS;
      end
```

Figure 10. An Algorithm of TS_OT Feeder/Master.

```
Algorithm TS_OT Processors
begin
1.      initialize as one dedicated partition space;
2.      read t, ParmSet from dedicated partition space;
3.      if receive command calculateIECMax from TS_OT Feeder through space based remoting;
4.          while (πw is not empty) do
5.            for (iteration Ij , j=j+1, …,15) do
6.              begin
7.                merge possible interaction element combination
        in IESet into complete test case, τ;
8.                calculate the current maxIEC highest interaction
       coverage;
9.                compare current maxIEC with previous maxIEC;
10.                 if (current maxIEC is higher than or equal to
           previous maxIEC)
11.                     stored current maxIEC and its test case , τ;
12.                else stored previous maxIEC and test case , τ;
13.               compare current maxIEC with pmaxIEC;
14.                if ( selected current maxIEC is higher than or
          equal to previous pmaxIEC
15.                     send current maxIEC and its test case , τ to
          TS_OT Feeder PU via a reducer
16.                else continue test case generation and search
           for best solution
17.             end
18.             send current highest maxIEC and its test case, τ
       to TS_OT Feeder PU via a reducer;
  end
```

Figure 11. An algorithm of TS_OT processor worker.

Table 5. Fixed p=10 and t=3 with varying v from 2 to 10.

Strategy	TS_OP		TS_OT	
Parameter value, v	Size	Time(s)	Size	Time(s)
2	16	2.44	18	5.31
3	66	12.92	67	63.98
4	156	36.23	167	326.43
5	308	110.48	327	1457.86
6	523	249.78	571	4147.34
7	825	397.02	1049	10619.49
8	1205	792.39	1412	27826.58
9	1703	1504.94	1966	53818.95
10	2307	2626.19	2768	102762.78

ANALYSIS ON TEST SIZE GROWTH AND TEST GENERATION TIME

The aim on this group of experiment is to evaluate the characteristic of both strategies in term of test size growth and the generation time on single machine environment. Three types of experiments are carried out to determine the characteristic of the test size growth and the generation time. Here, all the results are obtained using Windows XP, with 2.13 GHz Dual CPU, 4 GB RAM, and JDK 1.6 installed on it. It should be noted that the time is recorded in second.

INCREASING PARAMETER VALUE

This experiment is carried out to investigate the test size growth and the generation time as the number of parameter values increases from 2 to 10 with fixed parameter of 10 and fixed interaction strength of 3. The result of the generated test size and test suite generation time are shown on the Table 5.

From Table 5, the test sizes are plotted against the number of parameter values as given in Figure 12.

As illustrated in Figure 12, the generated test size using TS_OP strategy always smaller as compare to generated test size using TS_OT strategy. The generation time is plotted versus number of parameter values as shown in Figure 13. The test generation time was also faster for TS_OP strategy as compared to TS_OT strategy for all setting.

Referring to Figures 12 and 13, the curve fitting analysis is applied on both plotted figures and conclude that both test size and test generation time are proportional quadratically with the number of values in both strategies. Overall for varying the parameter value from 2 to 10, the TS_OP strategy showed a better result both in term of test size and test generation time as compared to TS_OT strategy.

INCREASING PARAMETER

In this experiment, the test size growth and test generation time is recorded as the number of parameters increases with fixed parameter value and fixed interaction strength. The input parameter consist of fixed parameter value, v=4 and interaction strength of t=3 with 6 to 15 parameters. The result for this experiment is shown in the Table 2. Using Table 6 data, the test size versus number of parameters graph is plotted as shown Figure 14. Here, TS_OP strategy always produces a smaller test size as compared to the TS_OT strategy.

The graph of the test generation time versus number of parameters is plotted as shown in Figure 15. Referring to Figure 14, we conclude that the test size grows logarithmically with increasing number of parameters in both strategies. From Figure 15, we note that test generation time grows quadratically with respect to logarithmic scale of parameters for TS_OP strategy and in TS_OT the test generation time increase more steeply as compared to TS_OP strategy. The test generation time for TS_OP strategy is always faster than TS_OT strategy in all tested input parameter with varying parameter. These results are as expected since the TS_OP have a lesser computing complexity while generating test case as compared to TS_OT.

In TS_OT, all parameter need to be consider while generating test case whereas for TS_OP, initially on portion of parameter need to be processed until full number of parameter. Here the difference between the TS_OP and TS_OT strategy in term of test generation is larger as parameter is increased. This occurs due to computing complexity for higher parameter.

INCREASING INTERACTION STRENGTH

In this computational simulation, the test size growth and test generation time is recorded as the number of interaction strength increases with fixed parameter and

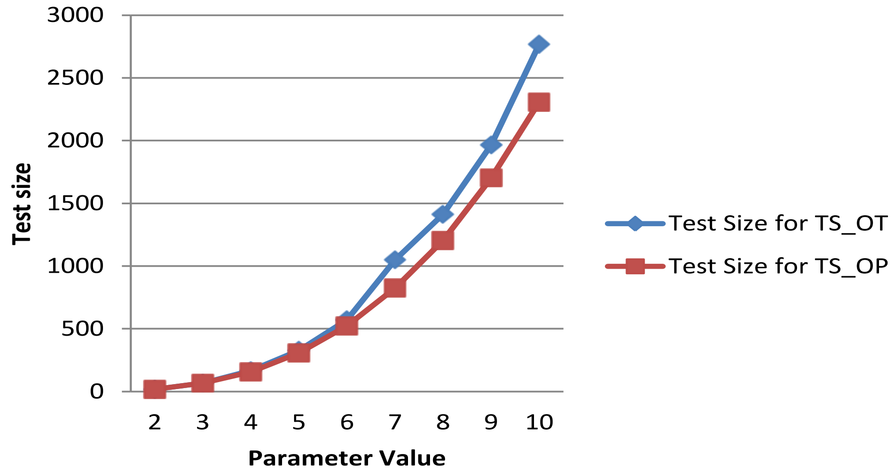

Figure 12. The test size growth for fixed parameter 10 and t=3 with varying parameter value from 2 to 10.

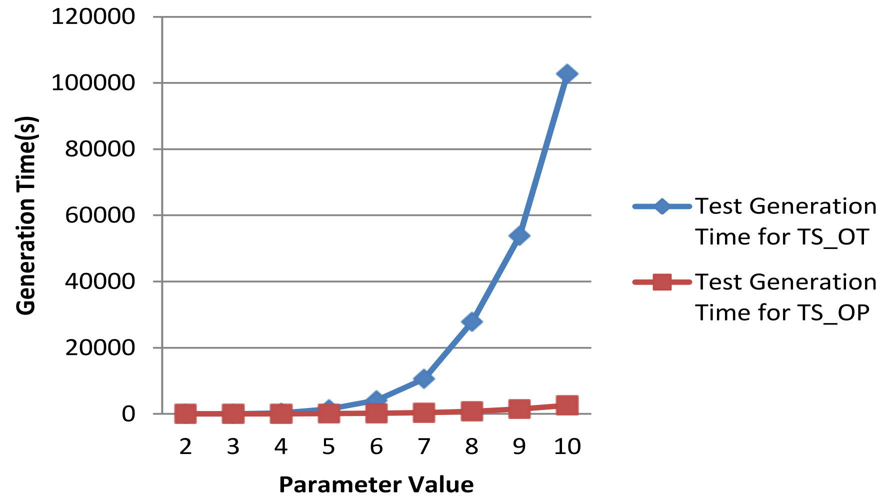

Figure 13. The generation time for fixed parameter 10 and t=3 with varying parameter value from 2 to 10.

Table 6. Fixed v=4 and *t*=3 with varying *p* from 6 to 15.

Strategy / Parameter, P	TS_OP		TS_OT	
	Size	Time(s)	Size	Time(s)
6	99	3.97	112	5.59
7	121	7.7	126	18.09
8	137	14.4	142	56.55
9	145	21.83	155	139.62
10	156	36.23	161	358.55
11	167	39.54	179	812.34
12	175	55.53	187	1457.5
13	180	83.17	194	2769.95
14	188	117.14	204	4675.42
15	194	158.8	217	8414.11

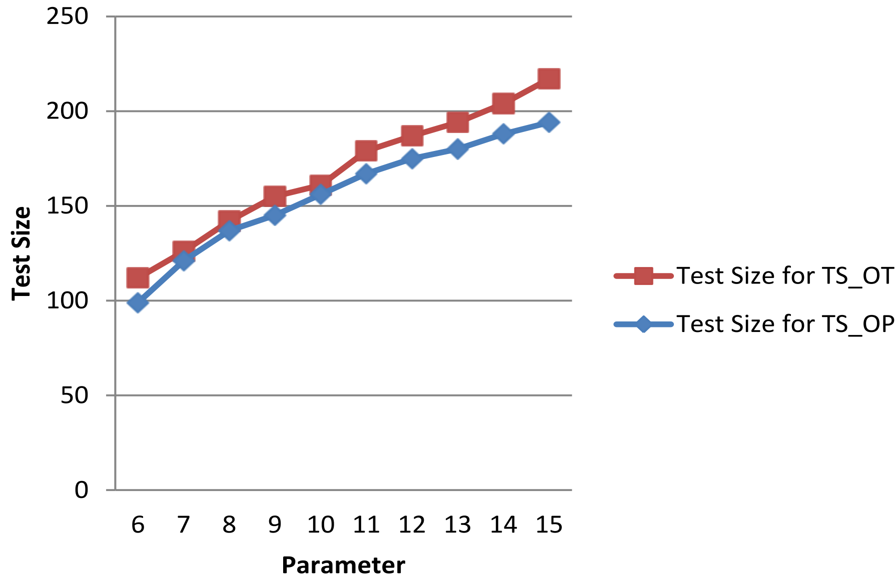

Figure 14. The test size growth for fixed parameter value, v=4 and t=3 with varying parameter from 6 to 15.

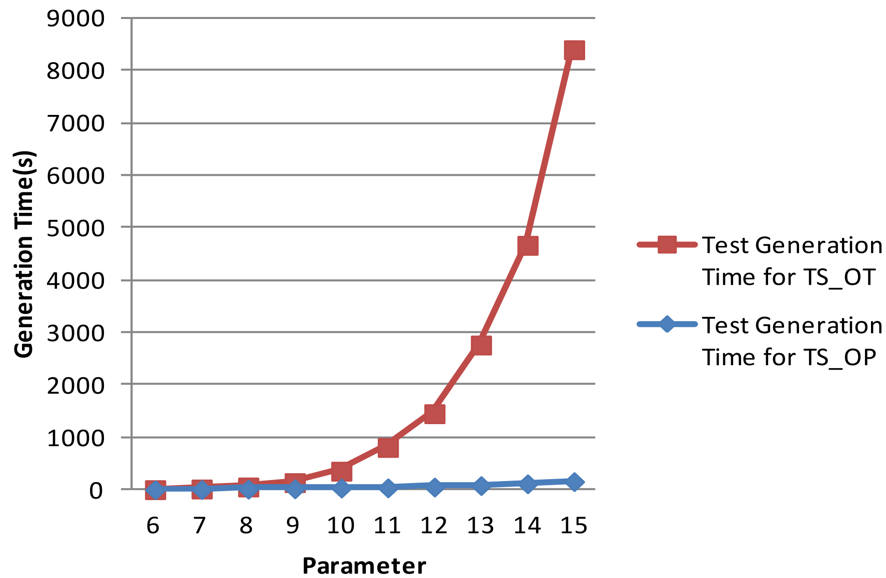

Figure 15. The generation time for fixed parameter value, v=4 and t=3 with varying parameter from 6 to 15.

their parameter value. The input parameter consist of fixed 10 parameter and fixed parameter value, v=3 with varying interaction strength from t= 2 to 7. The result is shown on the Table 7.

Referring to Table 7, the test size is plotted versus differences in term of test size are quite minimal and TS_OP is still able to generate smaller test suite size as interaction strength as given in Figure 16. Here, the compared to TS_OT. Similarly, the test generation time versus interaction strength is plotted as shown in Figure 17. From Figures 16 and 17, it is evident that the test size as well as test generation time grows exponentially as the interaction strength increases.

In both strategies, computational complexity increased rapidly as interaction strength is increased. However, due to less complexity in test suite generation in TS_OP, their

Table 7. Fixed *p*=10 and *v*=3 with varying *t* from 2 to 7.

Strategy	TS_OP		TS_OT	
Interaction strength, t	Size	Time(s)	Size	Time(s)
2	17	1.45	17	8.94
3	66	12.92	69	46.29
4	230	176.72	233	1344.04
5	732	3017.25	747	17329.28
6	2132	29500.67	2239	185253.06
7	5751	162350.93	6274	453833.16

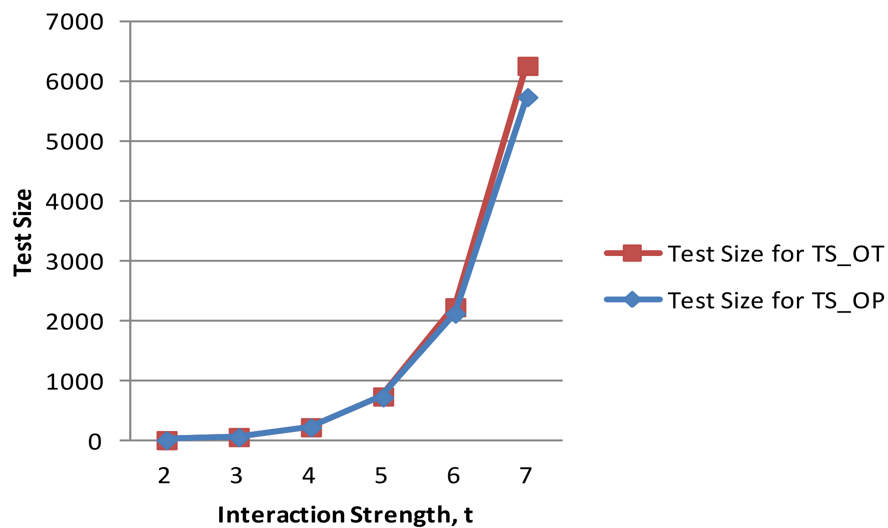

Figure 16. The test size growth for fixed parameter value, v=3 and parameter, p=10 with varying interaction strength from 2 to 7.

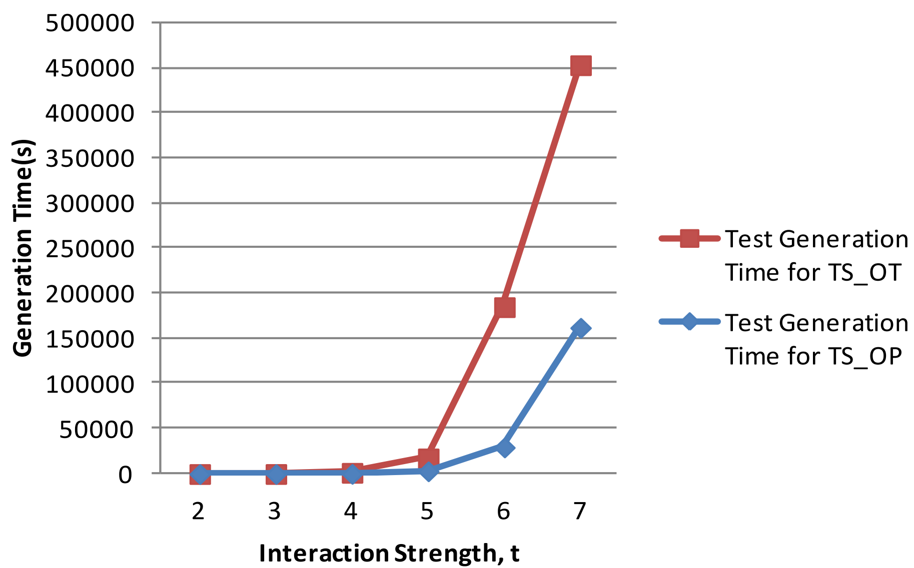

Figure 17. The generation time for fixed parameter value, v=3 and parameter, p=10 with varying interaction strength from 2 to 7.

Table 8. Uniform input parameter with fixed p=10, t=3 and v=10 on ten different machines.

Machine	Strategy							
	TS_OP		TS_OT		TS_OP	TS_OT	TS_OP	TS_OT
	Size	Time(s)	Size	Time(s)	Size ratio	Size ratio	Speedup	Speedup
1	2307	2626.19	2708	128110.06	0.996	0.907	1.00	1.00
2	2309	1732.28	2847	79525.88	0.996	0.954	1.52	1.61
3	2314	1537.06	3037	61738.09	0.999	1.017	1.71	2.08
4	2297	1467.53	2902	41857.83	0.992	0.972	1.79	3.06
5	2322	1323.7	2911	34435.28	1.003	0.975	1.98	3.72
6	2310	1244.83	3267	33453.11	0.997	1.094	2.11	3.83
7	2297	1236.25	3306	27921.41	0.992	1.107	2.12	4.59
8	2313	1194.36	3249	24839.53	0.999	1.088	2.20	5.16
9	2304	1231.06	3551	24627.84	0.995	1.190	2.13	5.20
10	2315	1246.62	2984	17445.19	1.000	1.000	2.11	7.34

generation time is faster compared to TS_OT strategy.

SCALABILITY ANALYSIS IN TERM OF SPEEDUP GAINED ON DIFFERENT MACHINE ENVIRONMENT

In the second group of experimentation for the speedup gained, the system consist of ten workstation interconnected through a Cisco switch with each machine running with a GigaSpaces middleware. The LAN speed is 100 M. Firstly, the scalability analysis is carried out in term of speedup gained while running on multiple machines environment. Speedup is deduced from execution time of single machine per execution time of multiple machines. Two experiments were carried out to determine the speedup gain from different input parameter:

1. Uniform input parameter of fixed parameter, p=10 with parameter value, v=10 and interaction strength of t= 3.
2. Mixed input parameter value of TCAS with interaction strength of t=4.

Both experiments were initially carried out on single workstation using GigaSpaces Service Grid and then on multiple workstation for up to 10 machines.

The result for uniform input parameter of fixed parameter, p=10 with parameter value, v=10 and interaction strength of t= 3 is shown on the Table 8 and for mixed input parameter value of TCAS with interaction strength of t=4 is shown in Table 9.

Both strategies inhibited a non deterministic nature which may result in different test size for each simulation run. These results recorded are best representative for the minimum number of test size of many simulations run, around 10 to 15 runs for each input parameter configuration setting. Referring to Tables 8 and 9, the differences in the test suite size for both, uniform and mixed input parameter value in TS_OP strategy is smaller than TS_OT strategy while running on different number of machine as indicated by the test size ratio. This happened due to different distributed strategy, in TS_OP strategy, the distribution of interaction element data based on the TS_OP Processor partition space's assigned parameter value and the current position of parameter that have been added to the test case whereas in TS_OT strategy, the distribution of interaction element using hash table routing mechanism to each TS_OT Processor partition space. This lead to a situation where the TS_OP strategy can produce an initial test case with maximum interaction coverage in both single or multiple machine environment whereas the TS_OT strategy can only produce an initial test case with maximum interaction coverage in single machine environment. Therefore, while running on multiple machine environments, the TS_OT can produce the test case by merging all possible interaction element in a dedicated partition space only which would not guarantee the production of an initial test case with maximum interaction coverage. As we partition the interaction element to higher number of physical machine, an initial test case produce will have less interaction coverage, therefore required bigger number of test case to cover all interaction element left in their partition space.

Overall, in term of test size optimality, the distributed strategy of TS_OP does not mortify the optimality of the test suite size while running in multiple machine environments. However, in TS_OT, the distribution strategy had lead towards less optimal test suite size when running on higher number of physical machine.

Using data from the Tables 8 and 9, we plot the speedup versus number of machine as shown in Figures 18 and Figure 19, respectively. In TS_OP strategy, for uniform input parameter and mixed input parameter, the increment of speedup value between 2 and 10 machines is almost similar as compared 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

Table 9. The mixed input parameter of TCAS value with interaction strength t=4 on ten different machines.

Machine	Strategy							
	TS_OP		TS_OT		TS_OP	TS_OT	TS_OP	TS_OT
	Size	Time(s)	Size	Time(s)	Size ratio	Size ratio	Speedup	Speedup
1	1355	5649.65	1565	112461.55	0.999	1.020	1.00	1.00
2	1349	2321.59	1590	64303.42	0.994	1.036	2.43	1.75
3	1370	2211.69	1597	36525.28	1.010	1.041	2.55	3.08
4	1366	2149.89	1597	26070.83	1.007	1.041	2.63	4.31
5	1371	2044.84	1595	22395.66	1.011	1.039	2.76	5.02
6	1365	1815.15	1624	18515.12	1.006	1.058	3.11	6.07
7	1347	1940.05	1660	14884.06	0.993	1.082	2.91	7.56
8	1349	2110.11	1565	17130.22	0.994	1.020	2.68	6.57
9	1362	1954.77	1655	12972.48	1.004	1.078	2.89	8.67
10	1356	2032.08	1534	13136.09	1.000	1.000	2.78	8.56

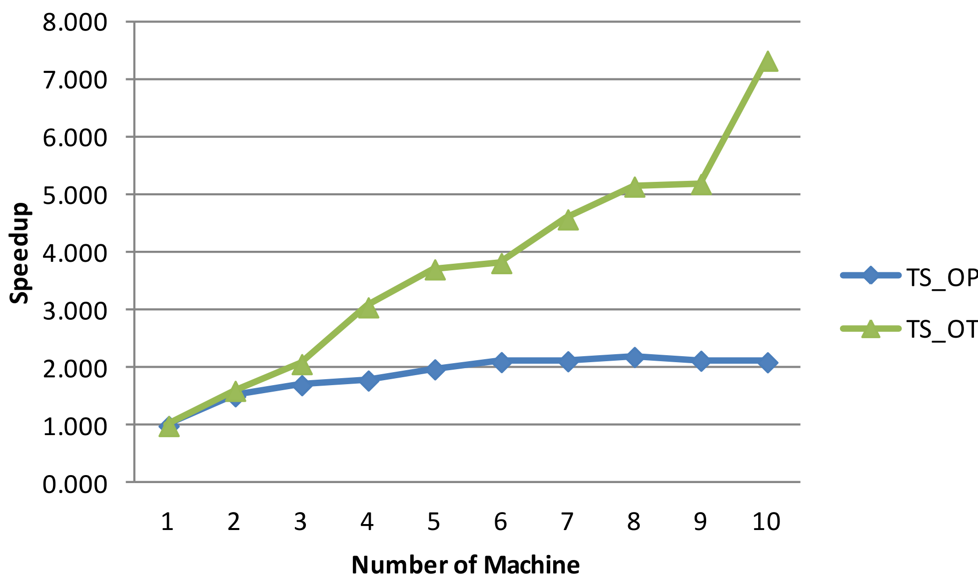

Figure 18. The speedup for fixed parameter value, v=10 and parameter, p=10 with interaction strength t=3 on 10 machines.

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 permitting faster computation in all available CPUs and resulted in shorter time for the test case generation. However, the scalability of TS_OP is dependent on the number of maximum parameter value. An increment of number of physical machine more than maximum parameter value will not be contributed for more speedup and in some cases slow down the test suite generation time.

As for TS_OT strategy, their distribution of interaction element among TS_OT Processors on different physical machine able to provide more memory space and computing power thus producing a faster test generation time while running in a multiple machine environment. The increase of physical machine almost give a linear speedup in term of test suite generation time. However, for some cases, the speedup dropped due to network latency but increase further on higher number of machine count.

Overall, it showed that distribution of computing work for test suite generation across multi machine environments always give a speedup as compared to single machine environment in both strategies. This points out the applicability of Tuple space technology as distributed shared memory platform in our implementation. Here, the

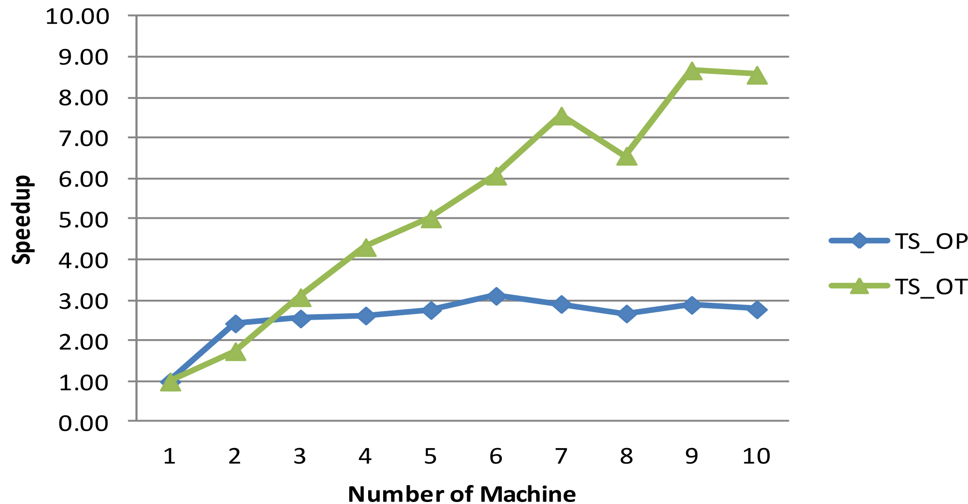

Figure 19. The speedup for TCAS value with interaction strength t=4 on ten different machines.

speedup gained in TS_OT strategy increased more linearly as the number of machine is incremented as compared to TS_OP strategy. Although the speedup gained in TS_OP do not increases linearly as compared to TS_OT strategy, the test generation time of TS_OP strategy still faster than TS_OT strategy in single or multiple machine environment settings.

COMPARISON WITH OTHER EXISTING STRATEGIES

In order to benchmark our approach, we also compare both strategies, TS_OP and TS_OT of the study with other well-known strategies. These strategies are MIPOG, IPOG, IPOG-D, IPOG-F, IPOG-F2, ITCH, Jenny TConfig, TVGII and GTWay. Here, the comparison aims to study the generated test suite size for both strategies, TS_OP and TS_OT against other strategies. To facilitate the comparison, we adopt a common configuration, TCAS module. The TCAS module is an aircraft collision avoidance system developed by the Federal Aviation Administration which has been used as case study in other related works (Lei et al., 2007; Kuhn and Vadim, 2006; Calvagna and Gargantini, 2009; Younis and Zamli, 2009; Younis et al., 2008; Lei et al., 2008; Younis and Zamli, 2010; Zamli et al., 2011). The module consist of 12 parameters consisting of two 10-valued parameters, one four-valued parameters, two three-valued parameters and seven two-valued parameters.

Both strategies, TS_OP and TS_OT is executed in our environment consisting of a desktop PC with Windows XP, 2.13 GHz Core 2 Dual CPU, 4 GB RAM. As for other strategies such as MIPOG, IPOG, IPOG-D, IPOG-F, IPOG-F2 ITCH, Jenny, TConfig, TVGII and GTWay, the generated test size result of all experiment is obtained from published result (Younis and Zamli, 2010; Zamli et al., 2011). By adopting the same input parameters and values, a fair comparison may be made between various strategy implementations in term of generated test size because the generated test suite size is not dependent on the system environments and specifications but rather on the algorithm of the strategy itself.

Table 10 depict the test size results obtained for the aforementioned experiments. The darkened cells with bold fonts indicate the best result for a specific input parameter configuration. The second best solution for that specific input parameter is indicated by the less darkened cell with *normal* fonts. Cells marked NA (not available) indicates that the results are unavailable due to lengthy test generation time.

Referring to Table 10, in comparison of our *t*-way "one-parameter-at-time" strategy, TS_OP to other parameter based such as MIPOG, IPOG, IPOG-D, IPOG-F and IPOG-F2 in term of generated test suite size, TS_OP always produce a satisfactory and competitive results and the strategy itself is only second to MIPOG and always outperforms others IPOG and its variant.

As a comparison of "one-test-at-time" strategy, TS_OT with others strategies in same group of test based such as ITCH, Jenny, TConfig, TVGII and GTWay in term of generated test suite size, TS_OT results is comparable or equal to TVGII results. GTWay always produce the most optimum results in this group of comparison and as for TS_OT, the test result is quite similar to GTWay for a small interaction strength of t=2 and 3. As we increased the interaction strength further to 4, 5 and 6, Jenny appeared to be second to GTWay and outperform others strategies such as TS_OT, TVGII and etc.

Overall, from the results in Table 10, the MIPOG strategy outperforms other strategies by producing the most optimum results in most of the configurations, while IPOG-D produce the worst results in all input configuration.

Table 10. The comparison of TS_OP and TS_OT with others strategies for TCAS value with varying interaction strength from 2 to 6.

Strategy, t	TS_OP	TS_OT	MIPOG	IPOG	IPOG-D	IPOG-F	IPOG-F2	ITCH	Jenny	TConfig	TVGII	GTWay
	Size											
2	**100**	101	**100**	**100**	130	**100**	**100**	120	106	**100**	101	**100**
3	408	403	**400**	**400**	480	402	427	2388	411	472	434	402
4	1355	1565	**1265**	1361	2522	1352	1644	1484	1527	1476	1599	1429
5	**4166**	4755	4196	4219	5306	4290	5018	NA	4680	NA	4773	4286
6	11105	12673	**10851**	10919	14480	11234	13310	NA	11608	NA	12732	11727

For interaction strength, t=2, all strategies except IPOG-D, TS_OT, ITCH, Jenny and TVGII produce a minimum test size of 100. As for TS_OP strategy, an optimum test suite size is produce for two inputs setting with interaction strength, *t*=2 and 5. FAs for other input setting of t=3, 4 and 6, the MIPOG strategy outperform others strategies in term of most optimum test size. However, the TS_OT strategy results are not as competitive and satisfactory as compared to TS_OP in comparison with other existing strategies.

Conclusion

In this paper, we developed, evaluated and compared two distributed strategy called TS_OP based on "one-parameter-at-a-time" and TS_OT based on "one-test-at-a-time" for *t*-way test suite generation on single and multiple machine environments using Tuple space technology.

The complexity analysis of test suite growth also indicated that the test size growth in the TS_OP and TS_OT strategy generally follows closely the combinatorial theory. The test suite size of same input parameter configuration generated by TS_OP strategy is smaller and more optimum than the test suite size generated by TS_OT strategy. Furthermore, the test generation time of same input parameter configuration using TS_OP strategy is faster than the test generation time using TS_OT strategy.

The scalability analysis also showed that distribution of computing work for test suite generation across multi machine environments always give a speedup as compare to single machine environment in both strategies. This points out the applicability of Tuple Space Technology as distributed shared memory platform in our implementation. Here, the speedup gained in TS_OT strategy increased more linearly as the number of machine is incremented as compared to TS_OP strategy but the test generation time of TS_OP strategy is still faster than TS_OT strategy in single or multiple machine environment settings.

Comparison between our both strategies, TS_ OP and TS_OT with existing strategies indicated that the test suite size produced by our TS_OP strategy is satisfactory and competitive in term of generating minimum test suite size. In the case where TS_OP is not the best, the test size is still within an acceptable value. As for the TS_OT strategy, the generated test size results are not as competitive and satisfactory as compared to TS_OP in comparison with other existing strategies. However, in comparison to its "one-test-at-a-time" group, TS_OT is quite competitive to best strategy within their group for a lower t of 2 and 3.

For future works, we would like to implement the TS_OP and TS_OT strategy on cluster machines with high RAM memory to minimise the network latency whilst generating the test suite.

ACKNOWLEDGMENT

This research is partially funded by the generous grants – "Investigating T-Way Test Data Reduction Strategy Using Particle Swarm Optimization Technique" from Ministry of Higher Education (MOHE), Malaysia and USM Research University Postgraduate Research Grant Scheme on "Distributed Strategy for Software and Hardware Test Planning Tool using Tuple Space Technology" from IPS, USM.

REFERENCES

Aguirre A, Borja R, Garciá C, Bracho-Rios J, Torres-Jimenez J, Rodriguez-Tello E (2009). A New Backtracking Algorithm for Constructing Binary Covering Arrays of Variable Strength. Advances in Artificial Intelligence, Springer Berlin Heidelberg, 5845: 397-407.

Bryce RC, Colbourn CJ (2007). The density algorithm for pairwise interaction testing. Software Testing, Verification Reli., 17: 159-182.

Bryce RC, Colbourn CJ (2009). A density-based greedy algorithm for higher strength covering arrays. Software Testing, Verification Reliability, 19: 37-53.

Burroughs K, Jain A, Erickson RL (1994). Improved quality of protocol testing through techniques of experimental design. ICC '94, SUPERCOMM/ICC '94, Conference Record, 'Serving Humanity Through Communications.' IEEE Int. Conf., 742: 745-752.

Calvagna A, Gargantini A (2009). IPO-s: Incremental Generation of Combinatorial Interaction Test Data Based on Symmetries of Covering Arrays. In Software Testing, Verification and Validation Workshops, ICSTW '09. International Conference. pp. 10-18.

Cohen DM, Dalal SR, Fredman ML, Patton GC (1997). The AETG system: an approach to testing based on combinatorial design. Software Engineering, IEEE Transactions, 23: 437-444.

Cohen DM, Dalal SR, Parelius J, Patton GC (1996). The combinatorial design approach to automatic test generation. Software, IEEE, 13: 83-88.

Cohen MB, Colbourn CJ, Ling ACH (2008). Constructing strength three covering arrays with augmented annealing. Discrete Mathematics, 308: 2709-2722.

Cohen MB, Dwyer MB, Shi J (2007). Interaction testing of highly-configurable systems in the presence of constraints. In Proceedings of the international symposium on Software testing and analysis ACM, London, United Kingdom. pp. 129-139.

Cohen MB, Gibbons PB, Mugridge WB, Colbourn CJ (2003). Constructing test suites for interaction testing. In Software Eng. Proc. 25th Int. Conf., pp. 38-48.

Czerwonka J (2006). Pairwise testing in real world practical extensions to test case generators. In Proceedings of 24th Annual Pacific Northwest Software Quality Conference, pp. 419-430.

Forbes M, Lawrence J, Kuhn R, Yu L, Kacker R (2008). Refining the In-Parameter-Order Strategy for Constructing Covering Arrays. J. Res. National Instit. Standards Technol., 113: 287-297.

GigaSpaces (2010). Website for GigaSpaces. http://www.gigaspaces.com.

Grindal M, Offutt J, Andler SF (2005). Combination testing strategies: a survey. Software Testing, Verification Reliability, 15: 167-199.

Hartman A, Raskin L (2004). Problems and algorithms for covering arrays. Discrete Mathematics, 284: 149-156.

ITCH (2010). IBM ITCH. 2010.

Jenny (2010). Jenny Web Page. 2010.

Kimoto S, Tsuchiya T, Kikuno T (2008). Pairwise Testing in the Presence of Configuration Change Cost. In Secure System Integration and Reliability Improvement, SSIRI '08. Second Int. Conf., pp. 32-38.

Klaib MF, Zamli J, K Z, Isa NAM, Younis MI, Abdullah R (2008). G2Way A Backtracking Strategy for Pairwise Test Data Generation. In Software Engineering Conference, APSEC '08. 15th Asia-Pacific, pp. 463-470.

Kuhn DR, Vadim O (2006). Pseudo-Exhaustive Testing for Software. In Software Engineering Workshop, SEW '06. 30th Annual IEEE/NASA, pp. 153-158.

Kuhn DR, Wallace DR, Gallo AM, Jr (2004). Software fault interactions and implications for software testing. Software Engineering, IEEE Transactions. 30: 418-421.

Kuhn R, Kacker R, Yu L, Hunter J (2009). Combinatorial Software Testing. Computer, 42: 94-96.

Kuhn R, Yu L, Kacker R (2008). Practical Combinatorial Testing: Beyond Pairwise. IT Professional, 10: 19-23.

Kuo-Chung T, Yu L (2002). A test generation strategy for pairwise testing. Software Engineering, IEEE Transactions. 28: 109-111.

Lei Y, Kacker R, Kuhn DR, Okun V, Lawrence J (2007). IPOG: A General Strategy for T-Way Software Testing. In Engineering of Computer-Based Systems, ECBS '07. 14th Annual IEEE Int. Conf. Workshops. pp. 549-556.

Lei Y, Kacker R, Kuhn DR, Okun V, Lawrence J (2008). IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing. Software Testing, Verification Reliability, 18: 125-148.

Lei Y, Tai KC (1998). In-Parameter-Order: A Test Generation Strategy for Pairwise Testing. In The 3rd IEEE International Symposium on High-Assurance Systems Eng. IEEE Computer Society.

Mandl R (1985). Orthogonal Latin squares: an application of experiment design to compiler testing. Commun. ACM, 28: 1054-1058.

McCaffrey JD (2009). Generation of Pairwise Test Sets Using a Genetic Algorithm. In Computer Software and Applications Conference, COMPSAC '09. 33rd Annual IEEE Int., 1: 626-631.

Meagher K, Stevens B (2005). Covering arrays on graphs. J. Comb. Theory Ser., B95: 134-151.

Nie C, Leung H (2011). A survey of combinatorial testing. ACM Comput. Surv., 43: 1-29.

Nurmela KJ (2004). Upper bounds for covering arrays by tabu search. Discrete Appl. Math., 138: 143-152.

Sampath S, Bryce RC, Viswanath G, Kandimalla V, Koru AG (2008). Prioritizing User-Session-Based Test Cases for Web Applications Testing. In Proc. Int. Conf. on Software Testing, Verification, and Validation. IEEE Computer Society.

Schroeder PJ, Bolaki P, Gopu V (2004). Comparing the Fault Detection Effectiveness of N-way and Random Test Suites. In Proceedings of the International Symposium on Empirical Software Engineering IEEE Computer Society. pp. 49-59.

Shiba T, Tsuchiya T, Kikuno T (2004). Using artificial life techniques to generate test cases for combinatorial testing. In Computer Software and Applications Conference, COMPSAC Proc. 28th Annual Int., 71: 72-77.

TVGII (2010). TVG Download Web Page.

Wenhua W, Yu L, Sampath S, Kacker R, Kuhn R, Lawrence J (2009). A combinatorial approach to building navigation graphs for dynamic web applications. In Software Maintenance, ICSM IEEE Int. Conf., pp. 211-220.

Williams AW (2000). Determination of Test Configurations for Pair-Wise Interaction Coverage. In Proceedings of the IFIP TC6/WG6.1 13th International Conference on Testing Communicating Systems: Tools Techniques Kluwer, B.V. pp.59-74.

Williams AW, Probert RL (2001). A Measure for Component Interaction Test Coverage. In Proceedings of the ACS/IEEE International Conference on Computer Systems Applications. IEEE Comput Soc., pp. 304-311.

Younis MI, Zamli KZ (2009). ITTW: T-way minimization strategy based on intersection of tuples. In Industrial Electronics & Applications, ISIEA IEEE Symposium, 1: 221-226.

Younis MI, Zamli KZ (2010). MC-MIPOG: A Parallel t-Way Test Generation Strategy for Multicore Systems. ETRI Journal, 32: 73-83.

Younis MI, Zamli K Z, Isa N (2008). A strategy for Grid based t-way test data generation. In Distributed Framework and Applications. DFmA First Int. Conf., pp. 73-78.

Yu-Wen T, Aldiwan WS (2000). Automating test case generation for the new generation mission software system. In Aerospace Conf. Proc., IEEE, 1: 431-437.

Zamli KZ, Klaib MFJ, Younis MI, Isa NAM, Abdullah R (2011). Design and implementation of a t-way test data generation strategy with automated execution tool support. Information Sciences, 181: 1741-1758.

Zamli KZ, Younis MI (2010). Interaction Testing: From Pairwise to Variable Strength Interaction. In Mathematical/Analytical Modelling and Computer Simulation (AMS), Fourth Asia Int. Conf., pp. 6-11.