Distributed t-way Test Suite Generation Algorithm for Combinatorial Interaction Testing

Zainal Hisham Che Soh, Mohammed I. Younis, Syahrul Afzal Che Abdullah, and Kamal Zuhairi Zamli
Software Engineering Research Group, Universiti Sains Malaysia
E-mail: {zainalhisham.ld07, myounis.ld0, syahrulafzal.ld07}@student.usm.my, eekamal@eng.usm.my

***Abstract*- Combinatorial testing is an effective technique to reveal faulty interaction inside a given software system by covering all possible interaction element combination of input parameters by at least one of generated test cases. The design of the algorithm is based on the replicated Master-Worker design pattern implemented using tuple space technology. The tuple space is a middleware that coordinates the communication between applications or process from heterogeneous platform by mean of exchanging tuples through a shared data space. In this paper, a new distributed algorithm for the construction of t-way covering test suites is presented, utilizing parallel processing for finding the maximum interaction coverage for each set of different worker test cases. Each worker sends one test case and maximum interaction coverage value to the master. The master selects one test case from all test cases sent by all workers based on highest number of interaction element covered into the final test suite. The master has the privilege to delete the interaction element that is already covered by selected test case in order to ensure optimal solution. The main goal of utilizing parallel processing is to increase the computing power and memory resources to overcome the combinatorial explosion problem for large input parameters and parameter values. Lastly, in this paper, the correctness of proposed algorithm is demonstrated using an illustrated example.**

***Keywords*- Combinatorial Interaction Testing, Worker-Master design pattern, Tuple Space Technology**

1. INTRODUCTION

Quality and reliable software system is really important in our daily life nowadays. To ensure quality, many test suites need to be tested to verify the software meet their specified requirement. However, when test suites become too large, they can be difficult to manage and expensive to run, especially when they involve complex software system and requires manual effort. In highly configurable software system, such as web application, the number of test cases can be enormous in testing all the possible interaction among the different input parameter of the whole system. In fact, due to its intrinsic complexity, modelling activities and formal specification of the whole system may require a great effort, thus require a lot of testing works. The unintended interaction among input parameter can remain undetected and can cause the whole software system failure.

To ensure detection of errors due faulty interaction among different input parameters, all possible interactions must be covered in the test suite. However, an exhaustive testing consideration of all possible test cases is not feasible solution due to limited resources. Therefore, a combinatorial testing approach can be used to ensure the interaction coverage whilst systematically minimise the number of test cases by generating a reduce list of test cases that required combination of

parameter values among t-way different input parameters to be covered at least one time in one of the generated test case. The pairwise or 2-way testing is one form of *t*-way combinatorial testing that is used to ensure the pairwise interaction coverage. There are a few existing sequential *t*-way CIT tool, which is an automated test suite generator used to generate the test suite such as AETG[3], IPOG[6], TConfig[5], IRPS[1], IPO[2], DDA[4] and G2Way[7]. Furthermore, to effectively detect and localize the error among the unintended interaction, it is necessary to enable higher strength of *t*-way interaction. For higher degree of strength of interaction testing, the number of test case may increase rapidly. Therefore, the generation of test suite of higher *t*-way interaction among input parameters will also consume more computational resources and can produces larger solutions.

For real world and complex software system, the possible size and combination of input parameters value are likely to be huge and can lead toward the combinatorial explosion problem in term of the number of interaction element to be covered for combination of parameter values during generating the test suite. Hence, the generation of test suite will require a lot of computing power and memory resources. Therefore, limited memory resources can lead toward out of memory and can cause the system to be hanged. In order to overcome the problem, we propose to utilize the parallel processing to generate the test suite to enable the delegation of the computing work and memory space.

In this paper, we present our approach in to combinatorial testing in designing a distributed algorithm to generate test suites for *t*-way testing for software system under test. The algorithm described in this paper is based on master-worker pattern on distributed tuple space shared memory framework. The remainder of the paper is organized as follows. Section 2 gives some insight on the topic and recently published related works. Section 3 describes our parallelization approach for TS-CIT and discusses several key design decisions. Section 4 presents demonstration of correctness in order to assess the validity of the proposed approach. Finally, section 5 draws our conclusions and point out the ideas for future extension of this work.

2. BACKGROUND

Early work on Combinatorial Interaction Testing exploits computational approaches. For instance, Automatic Efficient Test Generator (AETG) builds a test suite using greedy algorithm by iteratively adding one test case at a time until all the interaction element combinations of parameter values are covered [3]. In [2], the IPO strategy has been introduced, which builds a pairwise test suite for the first two parameters, extends the test suite to cover the first three parameters, and continues to extend the test suite until it builds a pairwise test suite for all the parameters. Heuristic search techniques usually start from a pre-existing test suite and then apply a series of transformations determine using a fitness function to the test suite until a complete test suite is reached that covers all the combinations.

Other than computational approaches, algebraic approaches have also been constructed. These approaches construct test suite using pre-defined rules using mathematical function such as Latin square, orthogonal array and graph theory. Other algebraic approaches are based on the idea of recursive construction, which allows larger test sets to be constructed from smaller ones for example TConfig [5].

Finally, computational and algebraic approaches have their own advantages and disadvantages such as computational approaches can be applied to any 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 the system configurations to which they can be applied.

3. THE DESIGN APPROACH FOR TS-CIT

We now describe about our work here on parallel algorithm for tuple space based combinatorial interaction testing (TS-CIT). The TS-CIT strategy distributes the computational power and memory into different workstations using tuple spaces. Tuple space technology enables communication between applications and devices in a network of heterogeneous computers by exchanging tuples[8]. The tuple space model comprises a conceptually shared memory store (called *tuple space*). The tuple space is accessed using simple operations such as read, write, take, scan and as well as more complex operations and database functionality. Input operations specify the tuple to be retrieved from the tuple space using a form of *associative addressing* in which some of the fields in the tuple (*template*) have their values defined. Application or services clients connected to shared data space are loosely connected and they have the freedom to attach or detach to the server at their own will. Clients can connect just long enough to send or receive messages only before disconnecting.

The design pattern used is master-workers paradigm. The master invoked all workers and passes information to connected workers regarding the parameter set, *ParmSet*; the strength of interaction coverage, *t* and interaction element group, *IEG* through shared data spaces. The worker process can be in a single machine or multiple machines (i.e. for scalability purposes) that connected to shared space. Each worker process takes one *IEG* and generates all possible interaction elements related to their assigned *IEG* and store it back into shared data space as *IESet*.

The master application generates all the exhaustive test case and delegate the test case equally among available workers to their local memory. Each worker determines the weight for their assigned set of test cases and send the maximum weight and their respective test case τ to data space and notify the master. When master receive all the notification from all connected worker, the master will select one test case with the maximum weight, *maxIEC* and load test case into latest test suite, *TS*. Master deletes the covered combination from shared data space and notifies the selected worker to delete selected test case in their local memory. All workers then recalculate their new maximum weight for available test cases and send it back the to data space. The generation of test suite will stop when the shared *IESet* is empty.

3.1. Master

In this section, we describe how the master application works:

a) The master invokes the all worker.

b) The master sends ParmSet, interaction strength *t*, and the IEG to shared data space.

c) The master generates the exhaustive test set and send by batch to respective worker local memory.

d) The master starts with an empty test suite (*TS*), and waits for all workers to send a notification on new test case τ and maximum interaction element coverage value *maxIEC* to shared data space.

e) The master reads the *maxIEC* value from each worker from shared data space. Then, the master chooses the test case τ corresponding to the highest *maxIEC* value to be added to latest test suite, *TS*.

f) The master issues command to the selected worker via shared data space to delete the selected test case τ from their own local memory and to delete interaction elements covered by in *IESet* and notify for recalculate new highest *maxIEC*.

g) The master checks if the *IESet* is empty or not. If empty, the master will consider the latest test suite, *TS* as the final generated test suite.

For clarity, the complete algorithm for master is given below:

Algorithm *TS-CIT 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, t* and *IEG* to shared data space;
4. start the workers to connect to shared data space
   and assign unique IEG to each workers;
5. sends by batch the generated test set to respective worker to their local memory;
6. **while** *(IESet is* not empty*)* **do**
   begin
7. wait for all worker send τ with *maxIEC* to shared data
   space*;*
8. read the *maxIEC* from shared data space;
9. choose the τ with highest *maxIEC;*
10. add *t* value to *TS;*
11. send delete command to on τ *of* highest *maxIEC;*
12. remove the covered interaction element in *IESet;*
13. send notify for recalculate of maxIEC to all worker;
14. **return** *TS;*
    end
end

3.2. Worker

In this section, we now describe how the worker works:

a) The worker first connects to shared data space and reads the ParmSet*, IEG* and *t* in shared data space.

b) Each worker reads the assigned *IEG* from shared data space.

c) The worker generates all possible interaction elements for their assigned *IEG* and send back to shared data space as *IESe*t.

d) Each worker reads all test case τ in their local memory and the worker determines the *maxIEC* of all τ and sends the highest *maxIEC* with τ to shared data space.

e) The worker reads the command from shared data space, if it contains delete command the worker deletes τ in their local memory and wait for notify recalculation of new *maxIEC* notification.

The complete algorithm for the worker is given below.

Algorithm *TS-CIT Worker (ParameterSet ParmSet,t)*
begin
1. connect to shared data space;
2. read *ParmSet, t* and their assigned *IEG* to shared data space;
3. generate all possible interaction element of their *IEG* and send into shared data space as *IESet*;
4. read the assigned generated test set in their local memory and their size ts;
5. **for** *(1..ts)* **do**
   begin
6. calculate τ with highest *maxIEC* to shared data space*;*
7. read command from shared data space*;*
8. **if** (delete command)
9. remove τ with highest *maxIEC;*
10. send τ with highest *maxIEC* to shared data space*;*
    end
end

4. DEMONSTRATES OF CORRECTNESS

Table 1 shows a configuration setting using four parameters and for parameter, A, sets with three values another parameters, B, C and D, set with two values. Lets A, B, C and D represent the four parameters and lower capital of parameters denotes their values. Here we illustrated a mixed value example to further emphasize on the flexibility of the selection of input configuration in our approach.

TABLE 1
FOUR PARAMETERS A, B, C AND D

Parameter value	**Input Parameter**			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
	a3			

Assume, we have 3 workers and 1 master connected to the shared data space. For the input parameter as in Table 1 and the

interaction strength *t*=3. Hence, for 3-way interaction strength, there are 44 interaction elements for the input parameter as shown in Table 2. These entire interaction elements are stored in shared data spaces as *IESet*. As for the exhaustive test case combinations would be $3^1x2^3 = 24$. The exhaustive test cases are divided to 3 workers in this example. Therefore each worker will have 8 test cases respectively. Let say test case 1 to 8 is for worker 1; test case 9 to 16 is for worker 2 and lastly test case 17 to 24 for worker 3 as shown in Table 3. Let us assume that all workers known that maximum interaction coverage, *maxIEC* for each test case is 4. Each worker concurrently calculate the maximum pair covered by each test case in their table and select the test case for first encounter of *maxIEC*= 4. If there are no *maxIEC* = 4, it will send the first encounter of *maxIEC*= 3 and so on. The worker shall delete the test case with maxIEC= 0 to minimize the search space of their available test cases. Each worker will send the selected test case and their *maxIEC* to master.

TABLE 2
ALL POSSIBLE INTERACTION ELEMENTS FOR 3-WAY INTERACTION FOR FOUR INTERACTION GROUPS

ABC-	AB-D	A-CD	-BCD
a1b1c1	a1b1d1	a1c1d1	b1c1d1
a1b1c2	a1b1d2	a1c1d2	b1c1d2
a1b2c1	a1b2d1	a1c2d1	b1c2d1
a1b2c2	a1b2d2	a1c2d2	b1c2d2
a2b1c1	a2b1d1	a2c1d1	b2c1d1
a2b1c2	a2b1d2	a2c1d2	b2c1d2
a2b2c1	a2b2d1	a2c2d1	b2c2d1
a2b2c2	a2b2d2	a2c2d2	b2c2d2
a3b1c1	a3b1d1	a3c1d1	
a3b1c2	a3b1d2	a3c1d2	
a3b2c1	a3b2d1	a3c2d1	
a3b2c2	a3b2d2	a3c2d2	

Master selects the highest *maxIEC* among the 3 connected workers. If all *maxIEC* is equal between them, it will randomly choose one. The selected test case is loaded into TS. Master will remove the interaction element covered from *IESet* and send a delete command to respective worker which their test case selected. Master also will notify all worker to start recalculate the *maxIEC* for remaining test case and it stops the generation of test suite when the *IESet* is empty.

TABLE 3
EXHAUSTIVE TEST CASE AND WORKER GROUP

Exhaustive Test Case	A	B	C	D	Worker Test Case
1	a1	b1	c1	d1	W1T1
2	a1	b1	c1	d2	W1T2
3	a1	b1	c2	d1	W1T3
4	a1	b1	c2	d2	W1T4
5	a1	b2	c1	d1	W1T5
6	a1	b2	c1	d2	W1T6
7	a1	b2	c2	d1	W1T7
8	a1	b2	c2	d2	W1T8
9	a2	b1	c1	d1	W2T1
10	a2	b1	c1	d2	W2T2
11	a2	b1	c2	d1	W2T3
12	a2	b1	c2	d2	W2T4
13	a2	b2	c1	d1	W2T5
14	a2	b2	c1	d2	W2T6
15	a2	b2	c2	d1	W2T7
16	a2	b2	c2	d2	W2T8
17	a3	b1	c1	d1	W3T1
18	a3	b1	c1	d2	W3T2
19	a3	b1	c2	d1	W3T3
20	a3	b1	c2	d2	W3T4
21	a3	b2	c1	d1	W3T5
22	a3	b2	c1	d2	W3T6
23	a3	b2	c2	d1	W3T7
24	a3	b2	c2	d2	W3T8

In this example, the number of iteration needed before the final test suite can be finalized is 13 as shown in Table 4. In the first iteration the worker 1 send test case W1T1, worker 2 send W2T1 and worker 3 send W3T1 immediately to master as all their maxIEC is equal 4. The W1T1 is selected by the master and add into TS. All interaction element covered by W1T1 which are a1b1c1, a1b1d1,

TABLE 4
ILLUSTRATION ON THE ITERATIVE PROCESS OF SELECTING TEST CASE BY 3 CONNECTED WORKER

Worker Test Case	Iteration of selecting one test case and no. of pair covered by worker test case												
	1	2	3	4	5	6	7	8	9	10	11	12	13
W1T1	4												0
W1T2	x	3	2	1	1	1	0	0	0	0	0	0	0
W1T3	x	3	2	2	1	1	1	1	0	0	0	0	0
W1T4	x	4											0
W1T5	x	x	3	3	3	2	1	1	0	0	0	0	0
W1T6	x	x	4	4	4	4							0
W1T7	x	x	x	X	x	X	4	4					0
W1T8	x	x	x	X	x	X	x	X	0	0	0	0	0
W2T1	4	3	3	2	1	1	1	1	1	1	0	0	0
W2T2	x	4	4										0
W2T3	x	x	x	4									0
W2T4	x	x	x	X	1	1	1	0	0	0	0	0	0
W2T5	x	x	x	X	4	3	3	3	3	***3***			0
W2T6	x	x	x	X	x	4	2	1	1	1	0	0	0
W2T7	x	x	x	X	x	X	3	2	1	1	0	0	0
W2T8	x	x	x	X	x	X	4						0
W3T1	4	3	3	3	3	2	2	2	2	1	1	0	0
W3T2	x	4	4	3	3	3	3	3	4				0
W3T3	x	x	x	4	3	3	3	3	3	3	***3***		0
W3T4	x	x	x	X	3	3	3	3	3	2	2	1	0
W3T5	x	x	x	X	4								0
W3T6	x	x	x	X	x	3	2	2	2	1	1	0	0
W3T7	x	x	x	x	x	3	3	3	2	2	2	1	0
W3T8	x	x	x	x	x	4	4	3	3	3	3	***3***	0

a1c1d1 and b1c1d1 are deleted from shared data space. The master also sends delete command to worker 1 to delete the W1T1 in its local memory. The master notifies all worker to recalculate the maxIEC for their remaining test case. In second iteration, the worker 1 sends test case W1T4, worker 2 sends W2T2 and worker 3 sends W3T2 to master. Master selects and loads W1T4 into TS and covered interaction element is deleted from *IESet*. As for iteration 3, the worker 1 sends test case W1T6, worker 2 sends W2T2 and worker 3 sends W3T2 and master selects test case W2T2 and loads into TS.

The iteration goes on and in iteration 7, the worker deletes the test case W1T2 because all their interaction element is already covered by other selected test case. Then in iteration 8, the worker 1 sends test case W1T7, worker 2 sends W2T5 and worker 3 sends W3T2 with respective *maxIEC* of 4, 3, 3. The master will select test case W1T7. In iteration 9, *maxIEC* for each test case is now 3. Here the worker 1 will delete test case W1T3, W1T5 and W1T8 because their interaction element already covered. Then, worker 2 sends W2T5 and worker 3 sends W3T2 to master. Master then takes the test case W3T2 as selected test case.

TABLE 5
THE FINAL TEST SUITE AND TOTAL INTERACTION ELEMENT COVERED

Iteration No.	Final Test Suite	*maxIEC* selected
1	a1b1c1d1	4
2	a1b1c2d2	4
3	a2b1c1d2	4
4	a2b1c2d1	4
5	a3b2c1d1	4
6	a1b2c1d2	4
7	a2b2c2d2	4
8	a1b2c2d1	4
9	a3b1c1d2	3
10	a2b2c1d1	3
11	a3b1c2d1	3
12	a3b2c2d2	3
Total IEC		44

The process of selecting the right test case will continue until all the interaction element in IESet is covered and deleted as in iteration 13. Finally, the master loads the final test suite with twelve test cases as shown in Table 5 below. Eight test cases cover four interaction elements and four covers three interaction element and total they have covered all 44. Therefore, all 44 interaction elements are covered.

5. CONCLUSION

In this paper, we have highlight a distributed algorithm for t-way combinatorial interaction testing by delegating the test cases among worker process to obtain the faster determination of maximum interaction coverage of the test case and demonstrates the correctness through an example of mixed value input configuration. For an optimum solution and fast generation time of test suite, there are a few rules can be employ such as whenever *maxIEC* found, the worker instantaneous sends the test case to master without to iterate through all available test case. Secondly, all test case with all theirs interaction element covered is deleted from worker memory. In the future, we hope will be able to implement the algorithms and make experiment on their effectiveness with regard to other tools.

ACKNOWLEDGMENT

This research is partially funded by the generous grants – "Investigating Heuristic Algorithm to Address Combinatorial Explosion Problem" from Ministry of Higher Education (MOHE), Malaysia.

REFERENCES

[1] M. I. Younis, K. Z. Zamli, and N. A. M. ISA, "IRPS - An Efficient Test Data Generation Strategy for Pairwise Testing ", in Proc. of the 12th Intl. Conf. on Knowledge-Based and Intelligent Information & Engineering Systems (KES2008), Zagreb, LNAI Vol. 5177, pp. 493-500.

[2] Y. Lei and K. C. Tai , "In-parameter-order: a test generation strategy for pairwise testing," *Proceedings of 3rd IEEE Intl. Conf. on High-Assurance Systems Engineering Symposium*, 1998, pp. 254-261.

[3] D. M. Cohen, S. R. Dalal, J. Parelius, G. C. Patton, "The Combinatorial Design Approach to Automatic Test Generation," *IEEE Software,* Vol. 13, No. 5, pp. 83-87, September 1996.

[4] R.C. Bryce and C.J. Colbourn, "The Density Algorithm for Pairwise Interaction Testing," *Journal of Software Testing, Verification, and Reliability*, 17(3): 159-182, 2007.

[5] A. W. Williams and R. L. Probert. A practical strategy for testing pair-wise coverage of network interfaces. *In Proc. of the 7th Intl Symposium on Software Reliability Engineering (ISSRE)*, White Plains, New York, 1996.

[6] Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG: A General Strategy for T Way Software Testing", in Proc. of the 14th Annual IEEE Intl. Conf. and Workshops on the Engineering of Computer-Based Systems, Tucson, AZ, March 2007, pp. 549-556.

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

[8] T. J. Lehman, A. Cozzi, Y. Xiong, J. Gottschalk, V. Vasudevan, S. Landis, P. Davis, B. Khavar, and P. Bowman. Hitting the distributed computing sweet spot with TSpaces. *Computer Networks*, 35(4):457–472, 2001.