

Voting Multi-Dimensional Data with Deviations for Web Services under Group Testing

Wei-Tek Tsai, Yinong Chen, Dawei Zhang, Hai Huang
Department of Computer Science and Engineering
Arizona State University, Tempe, AZ 85287-8809

Abstract

Web Services (WS) need to be trustworthy to be used in critical applications. A technique called WS Group Testing has been proposed which can significantly reduce the cost of testing and ranking a large number of WS. A main feature of WS group testing is that it is able to establish the test oracles for the given test inputs from multiple WS and infer the oracles by plural voting. Efficient voting of complex and large number of data is critical to the success of group testing. Current voting techniques are not designed to deal with such a situation. This paper presents efficient voting algorithms that determine the plural value on multi-dimensional data and large number of data. The algorithm uses a clustering method to classify data into regions to identify the plural value. Experiments are designed and performed to concept-prove the algorithms and their applications with group testing.

Keywords: *Web Services Testing, Group Testing, Voting, Clustering.*

1. Introduction

Web Services (WS) aim at making heterogeneous web-based applications interoperate [1], so that they share their business logic, data and processes through a program interface across a network. They allow different applications from different sources to communicate with each other without customized coding. WS use XML-based representation and communication and are not tied to any operating system or programming language [1-5].

WS are changing the way we develop and use computer software. They are based on the Service-Oriented Architecture (SOA), which causes a paradigm shift away from the traditional product-oriented software architecture and enables the development of applications from the view point of

services. New WS can be composed from existing WS following certain application semantics and composition syntax. A typical WS development process can be outlined as follows:

1. Define the services required;
2. Use the standard WSDL, or other specification languages such as OWL-S and DMAL-S, to specify the requirements;
3. Search the Web to determine if there exist WS that meet the specification;
4. If found, goto step 11;
5. Decide if the service is large enough to be decomposed into smaller sub services;
6. If it is not worthwhile to decompose, develop the WS and test using traditional software development process. Goto step 12
7. Decompose the service specification into the specifications of N sub services;
8. For each sub service, goto step 3;
9. If no satisfactory WS are found for certain sub services required, develop the sub services using traditional software development process;
10. Assume k_i WS are found or developed for the sub service i , there are $k_1 * k_2 * \dots * k_N$ possible composite WS;
11. Test all or a subset of the possible (composite) WS;
 - a. If satisfactory WS are found, choose the best WS that meet the requirements.
 - b. If not, develop the WS and test it in traditional way;
12. Register the WS to a service broker, e.g., a UDDI server, so the WS can be found and used by other service requestors.

As can be seen from the process, the construction of WS is different from that of the traditional software. The differences include:

- WS must be specified in a standard format using standard language so that they can be registered, tested, and used by other service requestors;

WSDL, OWL-L, and DAML-S are standardized languages for this purpose [6-7].

- WS must be registered so that it can be found by other service requestors; UDDI is used for this purpose [8-9].
- WS can be remotely invoked through stand protocol like SOAP [6, 10], so that WS requestors can access the WS over the Internet;
- Existing WS can be used to compose new WS by service providers. In this case, the service providers are service requestors because they request to access the existing WS. It is a business decision if to use (and thus to pay for) the existing WS, or to develop new WS following the same specification. The new WS can also be offered for public use (and thus create revenue). Once a decision is made to use the existing WS, the process outlined above can be used to find and test the existing WS.

As the result, hundreds and thousands of WS could be found that were developed following the same public specification. Now the question is, how can these WS be tested efficiently at runtime and possibly in real time. Efficiency is important because a new service may be needed immediately, if, for example, a critical service, or a component service, in use failed or become unavailable suddenly.

We have recently proposed efficient techniques that can group test large number of WS, automatically generate test cases, and automatically generate oracles [11]. The oracle generated by plural voting is reliable even if the WS are under group testing are not reliable with multiple faults.

The problems need to be addressed include:

- There are large number of WS with the same functions to test;
- The WS are developed by different providers and the source code may not available;
- The outputs of the WS may not be a numbers that can be simply compared;
- The outputs of the WS may not be judged simply as correct or incorrect. There is a margin of tolerance on the outputs of the WS.

Voting has been widely used to select correct results among multiple replicas since it was introduced by von Neumann in 1956 [12]. Many variations have been proposed in past half century. Lorzak summarized in [13] the existing voting schemes in four categories: majority voting, median voter, plural voting, and weighted averaging voting. The performance of the four voting mechanisms was studied based on different input space, which gave users the insight on selecting the best voting strategies for their own applications. Blough extended the voting schemes in [14-15] to include a probabilistic model

exploring the optimal voting strategies for fault-tolerant systems. Yacoub proposed in [16] a simulated approach to analyze the behaviors of a voting system, the reliability of the voting system, and the probability of reaching a consensus, assuming that each node cast one of the three votes: correct, incorrect, or abstain. The simulation enumerates all possible decisions in a decision tree and studies different effects. Bass compared in [17] different voting algorithms for redundant software design and reported that majority voting is efficient and reliable in finding correct results even if individual software components are not reliable with multiple design faults. This is because independent faults tend to manifest in different forms of errors and at different time.

In this paper, we proposed an innovative voting scheme that combines plurality voting and probabilistic voting on a multi-dimensional data, which can efficiently evaluate the outputs of large number WS under group testing. The voting mechanism can handle numerical, non-numerical, precise and imprecise (with tolerable deviations) outputs. The voting mechanism is based on majority principle and statistic deviations theory. The main idea is to sort the outputs into clusters according to the properties they exhibit, distinguished by their sizes (majority principle) and the standard deviations (deviation theory) among their members. Users can either interactively select the cluster containing the satisfactory WS or let the built-in learning mechanism to adjust weights between the impact of the size and the standard deviation.

The paper is organized as follows. Section 2 outlines the grouping test technique. Section 3 introduces the new voting mechanism in detail. Section 4 presents a case study and experiment results. Section 5 concludes the paper.

2. Group Testing

WS testing is different from traditional software testing and requires distributed and coordinated testing. It needs to handle a large number of WS at runtime and possibly in real time. The group testing technique was originally developed for testing a large number of blood samples. Blood group testing has been since applied in diverse areas primarily to identify defectives in a homogeneous pool of items in a manner more efficient than individual testing, including software regression testing [17]. WS group testing is different from blood group testing in many ways. Table 1 compares and contrasts the two group testing techniques.

We proposed a group testing-based technology called Adaptive Service Testing and Ranking with Automated oracle generation and test case Ranking

(ASTRAR) to test a large number of WS efficiently and at runtime [11]. ASTRAR can test a large amount of WS at both the unit and integration levels.

Table 1. Comparing and contrasting blood and WS group testing

Com-pared features	Blood Group Testing (BGT)	Web Services Group Testing (WSGT)
Testing goals	Find bad samples from a large pool of blood samples.	Rank WS in a large pool of WS with the same specification; Rank the fault detection capacity of test scripts; Determine the oracle of each test case; and Fault identification.
Optimization objectives	Minimize the number of tests needed.	Minimize the number of tests and voting needed.
Sample mix	Arbitrary and physical mix.	Interoperability is constrained by WSDL, DAML-S, OWL-S, and composition semantics such as ontology.
Testing methods	Bio/chemical tests.	WS unit, integration, and interoperability testing using adaptive, progressive, and concurrent testing.
Testing location	Centralized testing	Distributed and remote testing by agents and voters.
Verification	Contamination analysis.	Oracle comparison and/or majority voting
Test coverage	One test for each mix.	Need <i>many</i> tests for each group of WS to verify a variety of aspects.
Reliability evaluation	Reliability of testing process	Reliability of WS under test and testing process
Reliability of tests	Tests can be reliable or unreliable. Most BGT assumes tests are reliable.	The voting mechanism may be unreliable, and the number of faulty WS may be greater than the number correct WS to mislead the voter.

3. Voting Schemes for WS Group Testing

A complex WS can produce different kinds of outputs. It is not efficient to use a single method to vote on the different kinds of data. This section devises different voting schemes suitable for different kinds of data.

3.1. Equality Voting on Majority Principle

Majority-based quality voting is normally applied in the situation where enumerable numerical or non-numerical values are the outputs of the WS under test and the WS are expect to produce exactly the same output if they are designed correctly. When a group of WS with the same specification is tested, those WS are considered correct if their outputs agree with the majority outputs.

Non-numerical data is the data without numerical meaning, such as male and female, and blood type A, B, AB, and O. To perform efficient voting, the non-numerical data can be mapped to numerical data before voting. Obviously, a one-to-one mapping must be defined, for example, mapping female to 0 and male to 1, and mapping A, B, AB, and O to 0, 1, 2, and 3, respectively.

The enumerable data can be a structured data. In fact, many WS output an ordered list or multiple lists.

For example, a stockbroker WS may offer a ranked list of WS recommending the best buys of the stocks. A travel agency WS may offer a list of alternative flight tickets, a list of hotels, and a list of car rentals for a client to choose from. A simple algorithm to vote on such structured data is to compare each component value in the structure.

We have defined a two-phase voting scheme to improve the performance. Assume that the WS under test output k lists. The lengths of the k lists are stored in a k -element list. The voting scheme first votes on the list of the lengths. The lists whose lengths disagree with the majority are eliminated in the first round. The remaining lists are voted in the second round one by one.

This paper focuses on voting with deviation which is much more challenging and much more useful for most WS. Voting with deviation means that two WS can be both correct even if they produce different output. For example, two correct weather forecasting WS may forecast slightly different temperatures, because they use different climatologic models. The rest of section is devoted to voting schemes with deviations.

3.2. One-Dimensional Clustering Voting

This section assumes m WS are under group testing, each of which will output a simple numerical

data. Certain deviations among the data are allowed. The aim of voting is to find the most probable correct output (oracle) for each input and the most probable correct WS.

Assume the set of m WS under test is $\mathbf{W} = \{W_1, W_2, \dots, W_m\}$. When a test case is applied to all the WS, the corresponding output vector is $\{O_1, O_2, \dots, O_m\}$, which forms a one-dimension data array. The following two algorithms define the voting scheme:

Algorithm 1: Rank WS for a single test input.

1. Input: $\mathbf{O} = \{O_1, O_2, \dots, O_m\}$; // The outputs of the WS are the input to the voter
2. Sort the input vector to obtain a sorted vector: $\mathbf{O}' = \{O'_1, O'_2, \dots, O'_m\}$;
3. Compute the distances between adjacent elements in the sorted vector to obtain: $\mathbf{D} = \{D_1, D_2, \dots, D_{m-1}\}$, where $D_i = O'_{i+1} - O'_i$
4. Sort the distance vector in the descent order to obtain: $\mathbf{D}' = \{D'_1, D'_2, \dots, D'_{m-1}\}$;
5. For a given deviation δ , partition \mathbf{O} into a minimum number of clusters, so that the distances between the elements in each cluster is less than δ .
6. Find the largest cluster \mathbf{MaxO} and the mean value ε of the outputs among the elements in \mathbf{MaxO} ;
7. Re-sort the ranked \mathbf{O} : $\mathbf{O}'' = \{O''_1, O''_2, \dots, O''_m\}$, place O''_i before O''_j , if $|O''_i - \varepsilon| \leq |O''_j - \varepsilon|$
8. Output sorted WS: $\mathbf{W}'' = \{W''_1, W''_2, \dots, W''_m\}$, where, W''_i corresponds to O''_i .

The main idea of the algorithm is to find the largest group of WS, whose outputs are within the allowed deviation; use the mean output value of this group as the **oracle** (correct output); and sort the WS according to the distances between their outputs and the oracle. Since the distances have been sorted, finding the clusters can be done in linear time. Thus, the time complexity of the algorithm is determined by the complexity of the sorting algorithm, which is $O(m \log m)$.

Algorithm 1 ranks the WS in a single test. To rank the WS after n tests, the following algorithm can be used.

Algorithm 2: Rank WS over n test inputs

1. Set weight vector $\mathbf{T} = \{T_1, T_2, \dots, T_m\} = \{0, 0, \dots, 0\}$
2. For test case $i = 1$ to n do
3. Call algorithm 1;
4. Obtain ranking for test case i : $\mathbf{W}'' = \{W''_1, W''_2, \dots, W''_m\}$
5. For $j = 1$ to m do $T_j = T_j + \text{Rank}(W_j)$;

6. Sort the \mathbf{T} in ascending order.

7. Output the WS corresponding to the sorted \mathbf{T} values.

The main idea of the algorithm is to use the accumulative ranks of the WS over n tests to rank the WS. The complexity of the algorithm $O(n*m*\log m)$, because the algorithm 1 is called n times.

3.3. Multi-Dimensional Voting Algorithms

Last section discussed the one-dimensional clustering method. This section extends the method to a voting algorithm for n -dimension data.

Assume $\mathbf{W} = \{W_1, W_2, \dots, W_m\}$ is the set of WS under test. For a given test case, the corresponding output set is $\mathbf{O} = \{O_1, O_2, \dots, O_m\}$, where each element (object) O_i , $i = 1, 2, \dots, m$, is an array of k elements, which can be interpreted as a point in a k -dimensional space. The distance between two objects can be defined the Euclidean Distance:

$d(i, j) = \|O_i - O_j\|^2 = \sum (O_{ir} - O_{jr})^2$, where O_{ir} and O_{jr} are the r^{th} element in O_i and O_j , respectively.

The distance between object i and the zero $(0, 0, \dots, 0)$ could be defined as the value of i and denoted by $d(i)$. Once $d(i, j)$ and $d(i)$ are defined, the Algorithms 1 and 2 in section 3.2 could be directly applied as follows:

1. For each object i , compute $d(i)$;
2. Sort the objects by $d(i)$;
3. Compute the distance $d(i, j)$ between adjacent pairs;
4. Initially assume all objects belong to one cluster. If the distance between two objects is greater than the permitted deviation, use the point to split the current cluster into two clusters.
5. Find the largest cluster.

Now the question is, will above algorithms actually work for multi-dimensional data? The answer is no. Figure 1 shows an example of in which there are four two-dimensional objects. Obviously, the objects should be divided into two clusters: $\{1, 3\}$ and $\{2, 4\}$. However, following the algorithm above, the objects will be sorted into 1, 2, 3, and 4. Then the distances between adjacent pairs will be computed. Since $d(1, 2) > \delta$, $d(2, 3) > \delta$, $d(3, 4) > \delta$. The four objects will be divided into four clusters.

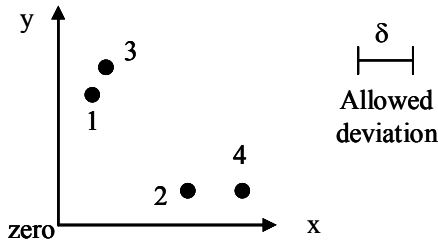


Figure 1. A counter example

This problem is caused by the fact that object that have the same or similar distances to the zero point can be far apart from each other. To address this problem, we propose a new algorithm that dynamically computes the geometric center of the cluster of objects and invite the object that is close to the center to join the cluster.

Algorithm 3: Clustering multi-dimensional objects

1. Input $\mathbf{O} = \{O_1, O_2, \dots, O_m\}$; // The outputs of the WS are the input to the voter
2. **SetClusters** = {};
3. If \mathbf{O} is empty, output ClusterSet, exit.
4. Randomly choose one object c to start a new cluster $\mathbf{C} = \{c\}$;
5. Compute the geometric center point u of the objects in \mathbf{C} ;
6. Computer the distance $d(u, v)$, where, $v \in \mathbf{O}$;
7. Find the object x with minimum distance to u : $d(u, x) \leq d(u, v)$;
8. If $d(u, x) \leq \delta$ (the allowed deviation) then move x from \mathbf{O} to \mathbf{C} : $\mathbf{C} = \mathbf{C} \cup \{x\}$ and $\mathbf{O} - \{x\}$;
9. Else a cluster is found: **ClusterSet** = **ClusterSet** \cup $\{\mathbf{C}\}$, goto step 3.

The geometric center point of a cluster is computed according to the average value of the corresponding objects. Figure 2 illustrates the execution process of Algorithm 3.

Assume there are six two-dimensional objects, given in the second column in the table in Figure 2. Column 3 lists the geometric centers. The first pair is the center point when there is only one object in cluster \mathbf{C} . The second pair, computed from the average values, is the center when there are two objects in the cluster, and so on. In the map in Figure 2, the six objects are marked by 1, 2, ..., 6. The center points are marked by the circled numbers.

After finding the clusters, the remaining steps will be similar to Algorithms 1` and 2.

Point number	Point (x y)	Central point	Central point number
1	(1 2)	(1.0 2.0)	
2	(3 1)	(2.0 1.5)	①
3	(3 5)	(2.3 2.7)	②
4	(5 2)	(3.0 2.5)	③
5	(2 6)	(2.8 3.2)	④
6	(4 7)	(3.0 3.8)	⑤

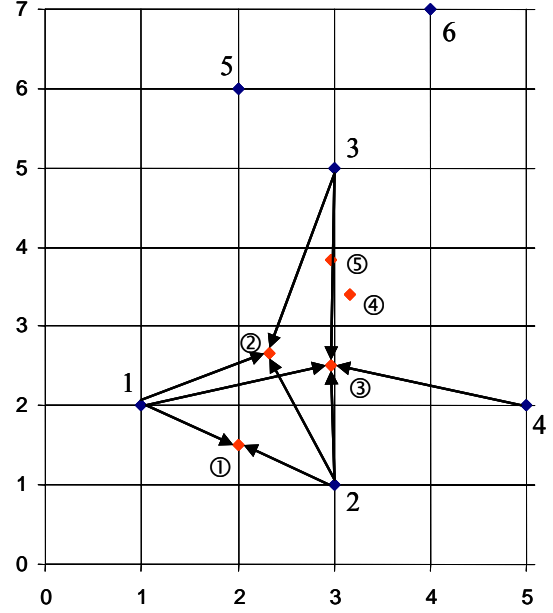


Figure 2. Geometric clustering

The complexity of the Algorithm 3 is given as follows: The loop (steps 3 through 9) may iterate m times, where m is the total number of objects. In step 5, computing the geometric center point needs to add the corresponding elements in all the objects in \mathbf{C} . It may take up to $k*m$ additions, where k is the dimension of the objects. This is a pessimistic estimation because the number objects in \mathbf{C} is not always m . It increases from 1 to m . It could be estimated to $k*\log m$. In step 6, computing the distance $d(u, v)$ may take $k*m$ operations. In total, the complexity is $(k*m + k*m)*m$, which is $O(k*m^2)$. Thus, Algorithm 3 is an efficient polynomial algorithm.

4. Case Study

WS predicting the best buys of stocks is a popular service offered by stockbrokers. To exercise our WS group testing and voting techniques, we have

developed 60 different WS, forecasting the stock prices using same predication models but different implementation. The assumption is, the majority of the WS will be implemented correctly and thus make similar predictions (within a given deviation) while the incorrectly implemented WS will generate different predictions in difference data range. In practice, we can use both self-developed WS and WS provided by other vendors and available over the Internet.

In the first experiment, we used the 60 WS to forecast the stock price of a single stock. We obtained the following 60 outputs as listed in the first four columns in table 2. In the next two columns, the oracle value and the outputs of the best 14 WS are computed and listed. The rank is based on the distance between the WS outputs to the oracle.

Figure 3 shows the clustering process. The curve marked by diamonds is the raw data outputs from the 60 WS. The second curve marked by squares is the data after being sorted in ascending order. The third curve marked by triangles is the differences between the adjacent pairs. Assuming the allowed deviation is 5% of the overall average, the algorithm finds four split points at 2, 8, 19, and 43, and thus dividing the data into five clusters. The largest cluster contains 42-

18 = 24 data. The average value among the 22 data is 20.81, which is then defined as the oracle. The 14 short-listed WS outputs are also highlighted in the first four columns in table 2.

Table 2. Experiment data: stock prices from 60 WS, oracle, and the best 14 WS

Stock Price	Stock Price	Stock Price	Stock Price	Oracle = 20.81	
				Rank	Best Value
19.8	10.36	21.5	19.8	1	20.77
9.15	8.24	21.71	29.8	2	20.72
12.24	9.80	30.73	29.15	3	20.93
29.3	11.04	20.62	32.24	4	20.93
13.31	14.03	19.41	29.3	5	20.62
30.35	6.41	20.47	13.31	6	21.12
8.17	20.93	18.58	20.35	7	21.12
11.49	20.72	22.56	28.17	8	20.47
12.13	21.75	20.28	31.49	9	20.41
9.2	20.77	21.99	32.13	10	20.35
30.85	22.10	21.12	29.2	11	20.28
15.22	21.12	20.41	28.85	12	21.35
8.41	20.39	20.93	15.22	13	21.50
2.75	21.50	21.35	28.41	14	21.71
14.67	21.71	22.34	22.75		

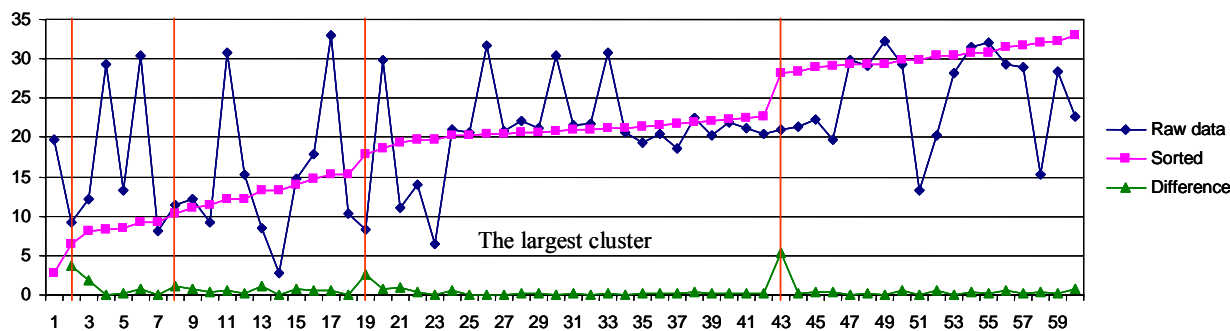


Figure 3. One-dimensional clustering

Another experiment is performed that outputs a two dimensional data (Stock_Price, Stock_Rank). Figure 4 shows the data and the clustering process. Each dot in the map is a pair of data. A circle is a cluster and the number in the circle is the order. The dot in the center of the circle is randomly picked up for starting a new cluster. The radius of the circle is the allowed deviation. The shadowed circle is the cluster containing most data. Because the data are widely scattered, the largest cluster does not contain 50% of the dots and thus the oracle is not established for this set of data.

5. Summary and Conclusion

This paper presented two majority voting algorithms, one for one-dimensional data and the other for multi-dimensional data. The voting algorithms are based on for a WS group testing framework which provides WS testing, reliability evaluation, and ranking services. The proposed algorithms can be used to rapidly evaluate a large number of WS at run time and in real-time. The challenges in designing these voters are that the voting mechanisms must be efficient and the voter must allow variations among the outputs of different WS. Both algorithms can handle data with permitted deviation and both have polynomial complexity. Although the multi-dimensional voting

algorithm can be applied to vote one-dimensional data, it is less efficient, compared to the one-dimensional

algorithm, if data is one dimensional. Experiment results show that these algorithms are effective.

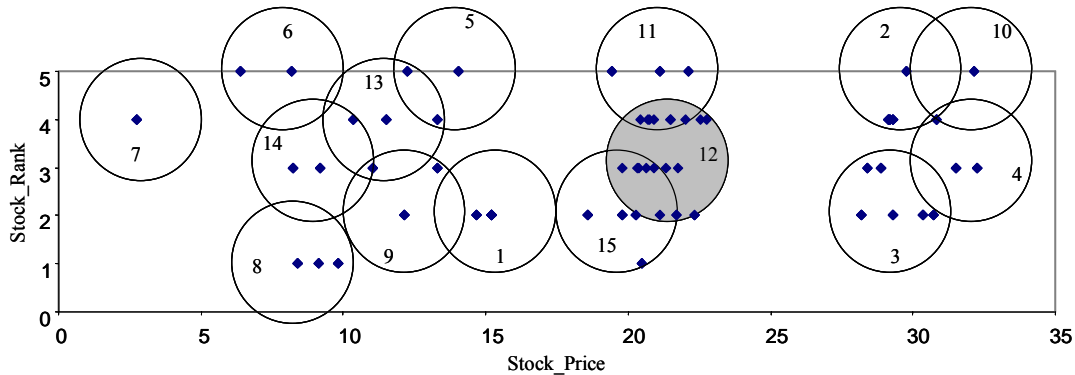


Figure 4. Two-dimensional clustering

References

- [1] B. De, "Web Services - Challenges and Solutions," WIPRO white paper, 2003, <http://www.wipro.com>
- [2] J. Bloomberg, "Web Services Testing: Beyond SOAP," ZapThink LLC, Sep 2002, <http://www.zapthink.com>
- [3] N. Davidson, "Testing Web Services," <http://www.webservices.org/>, in October 2002.
- [4] <http://www.digits.com/articles/web-hosting--what-is-a-web-service.htm>
- [5] http://searchwebservices.techtarget.com/ateQuestionResponse/0,289625,sid26_cid618845_tax288978,00.html
- [6] Uche Ogbuji, "Using WSDL in SOAP Applications," <http://www-106.ibm.com/developerworks/library/ws-soap/>, Nov, 2000
- [7] W. T. Tsai, R. Paul, Y. Wang, C. Fan, and D. Wang, "Extending WSDL to Facilitate Web Services Testing," Proc. of IEEE HASE, 2002, pp. 171-172
- [8] Doug Tidwell, "UDDI4J: Matchmaking for Web services," <http://www-106.ibm.com/developerworks/library/ws-uddi4j.html>, Jan 2001
- [9] W. T. Tsai, R. Paul, Z. Cao, L. Yu, A. Saimi, and B. Xiao, "Verification of Web Services Using an Enhanced UDDI Server," Proc. of IEEE WORDS, 2003, pp. 131-138
- [10] J. Myerson, "Testing for SOAP Interoperability," <http://www.webservicesarchitect.com/>, Feb 2002.
- [11] W. T. Tsai, Y. Chen, Z. Cao, X. Bai, H. Huang, R. Paul, "Testing Web Services Using Progressive Group Testing," Advanced Workshop on Content Computing, Zhenjiang, China, November 2004, pp.314-322
- [12] J.von Neumann, "Probabilistic Logics and the synthesis of reliable organisms from unreliable components," Automata Studies (Annals of Mathematics Studies,n34) 1956, pp43-98
- [13] P. R. Lorzak, A. K. Caglayan, "A Theoretical Investigation of Generalized Voters for Redundant Systems," Proc, Int'l Symp, Fault-Tolerant Computing, Chicago, June 1989, pp 444-451
- [14] D M. Blough, G. F. Sullivan, A Comparison of Voting Strategies for Fault-Tolerant Distributed Systems, Proc, 9th Symp, Reliable Distributed Systems, 1990 Oct, pp 136-145
- [15] D. M.Blough, G. F. Sullivan, Voting Using Predispositions, IEEE Transactions on Reliability, Vol 43, No. 4, 1994 pp604-616
- [16] S. Yacoub, "Analyzing the behavior and Reliability of voting Systems Comprising tri-state units Using Enumerated Simulation," Reliability engineering and System Safety, Vol 81, n2, 2003, pp133-145
- [17] J.M. Bass, G. Latif-Shabgahi, S. Bennett, "Experimental Comparison of Voting Algorithms in Cases of Disagreement," 23rd EUROMICRO Conference '97 New Frontiers of Information Technology, Budapest, September, 1997, pp.516 - 523.
- [18] D. Z. Du and F. Hwang, Combinatorial Group Testing And Its Applications, World Scientific, 2nd ed., 2000.