WS-DREAM: A Distributed Reliability Assessment Mechanism for Web Services

Zibin Zheng, Michael R. Lyu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Hong Kong, China
{zbzheng, lyu}@cse.cuhk.edu.hk

Abstract

It is critical to guarantee the reliability of service-oriented applications. This is because they may employ remote Web Services as components, which may easily become unavailable in the unpredictable Internet environment. This practical experience report presents a Distribute REliability Assessment Mechanism for Web Services (WS-DREAM), allowing users to carry out Web Services reliability assessment in a collaborative manner. With WS-DREAM, users in different geography locations help each other to carry out testing, and share test cases under the coordination of a centralized server. Based on this collaborative mechanism, reliability assessment for Web Services in real environment from different locations of the world becomes seamless. To illustrate the advantage of this mechanism, a prototype is implemented and a case study is carried out. Users from five locations all over the world perform reliability assessment to Web Services distributed in six countries. Over 1,000,000 test cases are executed in a collaborative manner and detailed results are provided.

1. Introduction

In general, Service Oriented Applications are built on top of Web Services which have standardized interface, loosely-coupled structure and cross-platform characteristics. In contrast to implementing all components from scratch for an application, it is much more efficient and economical to engage existing Web Services as components. Since remote Web Services may easily become unavailable in the unpredictable Internet environment, it is difficult to guarantee the reliability of applications developed on these Web Services.

Web Services reliability assessment techniques are therefore critical for establishing trustworthy Service-Oriented Applications [1,2]. Such assessments allow suitable Web Services be identified for applications from various Web Services available in Internet. Also, assessments are needed to select out optimal replication strategies for applications from a good deal of fault tolerance replication strategies [3,4], which employ identical or similar Web Services to enhance the application reliability. To perform accurate Web Service assessment, it is critical to design good test cases and conduct the testing in real-world experiments. Since most Web Service applications will be deployed to different locations in the world, it is important to carry out reliability assessments from these locations through various Web environments. Obtaining accurate assessment is therefore a formidable challenge for both Web Service users as well as Web Service providers.

To address this challenge, this practical experimental report presents a Distributed REriablity Assessment Mechanism for Web Services (WS-DREAM). WS-DREAM employs the concept of *user-collaboration*, which is an important concept contributing to the recent success of BitTorrent [5], and Wikipedia [6]. With WS-DREAM, users in different geography locations help each other to carry out testing under the coordination of a centralized server, which makes a distributed assessment of Web Services much easier. Moreover, the users can contribute individually-designed test cases to WS-DREAM, achieving a powerful, full-scale automated Web testing oracle. Under this collaboration manner, those users who plan to make assessment to the same Web Service will benefit from the intelligence (due to their individual design of test cases) of each other. They also benefit from the cumulated intelligence of prior users who have performed the Web assessment before.

WS-DREAM aims at two kinds of target users: Service users and Service providers. For Service users, instead of conducting time-consuming assessment themselves, they can make accurate reliability assessment of target Web Services by the facility of WS-DREAM. For Service providers, WS-DREAM can be employed to simulate real usage condition to their Web Services from various geographic locations via real Internet environment. This enables accurate real-life testing which is not available in the lab testing environment.

Design and implement of WS-DREAM will be presented. Also, to illustrate the functionality and applicability of WS-DREAM, users from five locations all over the world carry out assessment to Web Services located in six countries, and more than 1,000,000 test cases are executed and analyzed. The detailed results will be reported and discussed.

This paper is organized as follows: Section 2 introduces the architecture of WS-DREAM. Section 3 presents details of implementation, and practical experiment. Section 4 discusses experimental results, and Section 5 concludes the paper.

2. WS-DREAM Architecture

WS-DREAM includes a centralized server with a number of distributed clients. WS-DREAM server serves as a coordinator for the users. It is in charge of receiving test requests, creating test cases, scheduling test tasks, and analyzing test results. Distributed clients are running at computers of the user side. They carry out testing in a collaboration manner. As shown in Figure 1, steps of running WS-DREAM are as follows:

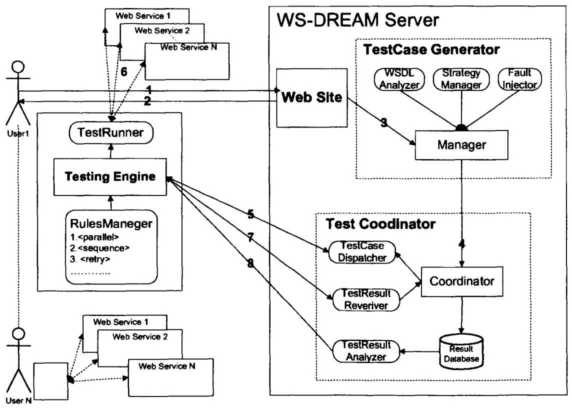

Figure 1. WS-DREAM Architecture

Step1: Users go to the Web site of WS-DREAM and submit assessment requests with the related information (i.e., target Web service address, timeout threshold, particular test cases, preferred replication strategies, etc).

Step2: The client-side of WS-DREAM is loaded and executed in the user's computer.

Step3: *TestCase Generator* in the WS-DREAM server creates test cases based on the interface of the target Web Service (WSDL file), test cases provided by users, and accumulated test cases in WS-DREAM. Fault injection techniques [7,8] are employed to create various fault-trigging test cases in addition to normal test cases. Test plans, which contain several test cases and a running rule, are composed.

Step4: *Test Coodinator* schedules testing tasks based on the number of current users and test requests.

Step5: The distributed client-side computer sends requests to the WS-DREAM server to get the test plan.

Step6: The user computer at the distributed client-side calls a *RulesManager* to resolve test plan and carries out testing to remote Web Services following the specified rules in the test plan.

Step7: The client-side computer sends back test results to the server, and repeats steps 5, 6 and 7 to execute more test plans.

Step8: After the test is completed, the WS-DREAM server engages *TestResult Analyzer* to process the collected data and send back detailed results to the user who submitted this assessment request.

After getting the analysis result, the user can stop the WS-DREAM client-side execution, or keep it open for running test plans for other WS-DREAM users.

2.1. Scheduling Algorithm

The dynamic changes of the current user number and test plan number in WS-DREAM makes the test tasks assignment difficult. Therefore, a scheduling algorithm is critical for enabling test plans to be executed in a collaborated and distributed manner. The following principles are taken into consideration when designing the scheduling algorithm:

- **Fairness.** Different Web Services should have fair chances to be assessed.
- **Distributed.** Web Services should be assessed by users in as many geography locations as possible.
- **Feasible**. Task assignment should dynamically adjust to the frequently changed number of users and number of test plans.
- **Efficient.** The algorithm should be efficient and it should not slow down the testing progress.

As shown in Figure 2, a Round-Robin based algorithm is designed to meet the above requirements. A client gets only one test plan from the server each time, and will send back the result to the server after executing the test plan. This design minimizes the influence of the sudden departure of a user, since at most one test result would be lost.

```
Begin
Let $T = \{T_1, T_2, ...., T_n\}$ be n WS test plan queues
    $i = \{i_1, i_2, ...., i_n\}$ be n indexes for T
    $c = \{c_1, c_2, .., c_m\}$ be m clients
    $j^x = \{j_1^x, j_2^x, ...., j_n^x\}$ be n counters for $c_x$
Set $i_1, i_2, ...., i_n$ to be 0
Repeat
    Wait for request from clients.
    $c_x$ = incoming client.
    If($c_x$ ==new client) set $j_1^x, j_2^x, ...., j_n^x$ to 0
    Else get $j_1^x, j_2^x, ...., j_n^x$ from the client.
    $k^x = \{i_1 + j_1^x, i_2 + j_2^x, ...., i_n + j_n^x\}$
    $m = i_y + j_y^x = Min(k^x)$
    If (multiple m) random select one
    Dequeue a testplan from $T_y$
    $i_x$++
    $j_x^x$++
    Return the test plan to the $c_x$
Until $i_x$ >= Max
End
```

Figure 2. Scheduling Algorithm

2.2. Replication Strategies

The abundant resources of identical and similar Web Services available in Internet make fault tolerance by design diversity [9] a natural choice for Service Oriented Applications to enhance reliability. The most suitable strategy for a Web Service application may be different depending on different application requirements and network conditions. WS-DREAM can be flexibly engaged to evaluate the performance of different strategies and assist developers to select out the most suitable strategy. 9 strategies are employed in WS-DREAM, which are shown in the following:

1. Parallel. The service oriented application sends requests to different Web Service replicas in parallel. The first properly returned response will be taken as the final outcome. This strategy can be used to tolerate faults and achieve better respond time performance. However, it consumes more computing and networking resource.

2. Retry. The same Web Service will be retried if it fails. This strategy can be employed to mask temporal faults.

3. RB. Another backup Web Service will be tried sequentially if the primary Web Service fails. In erroneous environment, the response time performance of RB is not good, since the backup Web Service is tried sequentially.

4. Parallel +Retry. The whole parallel block will be re-executed if fails.

5. Retry+ Parallel. Individual replica in the parallel block will be retried if the replica itself fails.

6. Parallel +RB. As shown in Figure 3, another parallel block using other replicas will be tried if the first parallel block fails.

7. RB+ Parallel. As shown in Figure 3, a replica in the parallel block will try another replica sequentially if it fails.

8. Retry+RB. A replica will retry itself first if it fails for certain times. Then another replica will be executed.

9. RB+Retry. A replica will try another standby replica first if it fails. After trying *n* times without success, the whole RB process will be retried.

XML-based scripts are used to express strategies in WS-DREAM; therefore, new strategies can be easily added. Figure 3 shows the expression samples of strategies type 6 and type 7.

```
<sequence>
  <parallel>
    <id>A1</id>
    ............
    <id>An</id>
  </parallel>
  <parallel>
    <id>B1</id>
    ............
    <id>Bn</id>
  </parallel>
</sequence >
```

```
<parallel>
  <sequence>
    <id>A1</id>
    <id>B1</id>
  </sequence >
  <sequence>
    <id>A2</id>
    <id>B2</id>
  </sequence >
  ............
</parallel>
```

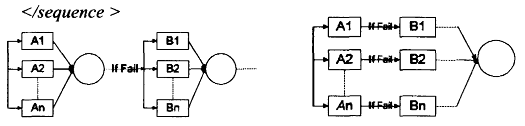

Figure 3. Expressions of Type 6 and Type 7

3. Implementation and Experiments

Based on the architecture design in Figure 1, a prototype of WS-DREAM is implemented using Java and Axis. The client-side of WS-DREAM is realized as a signed Java Applet, which is run with Internet

Browsers at the user computers. This provides a convenient way for users to carry out testing seamlessly, as Internet Browsers will load and update the client-side executions automatically.

The server-side of WS-DREAM includes several components: an *HTTP Web site* (running on Apache HTTP Server), a *TestCaseGenerator* (Java project using JDK6.0 and Axis library), and a *TestCoodinator* (Java Servlet running on Tomcat 6.0). MySQL is used to record testing results. These server-side components are loosely-coupled. Design diversity can be easily employed to enhance reliability.

Practical experiments are conducted in real Internet environments to illustrate WS-DREAM in real-world executions. We assume a user named Ben employs WS-DREAM for reliability assessment on several target Web Services, which will be employed in his commercial Web site. These Web Services include: six identical Amazon book displaying and selling Web Services distributed in six countries, a Global Weather Web Service to display currently weather information, and a GeoIP Web Service to get geography information of Web site visitors. Users of WS-DREAM, who are from five locations all over the world, perform this assessment in a collaborative manner. Over 1,000,000 test cases are executed.

In the above example, the *timeout* period is set to be 10 seconds by Ben. If a Web Server does not respond within the 10-second period, the request will be terminated and a failure is recorded. In practice, the value of timeout threshold is application-dependent and can be set by users in WS-DREAM based on the need of their applications. Detailed information of test cases, test plans, and test results of this example is available in the WS-DREAM Web site [10].

WS-DREAM is therefore employed by Ben for following purposes:

1) Assess the reliability of the target Web Services.
2) Measure the performance of different replication strategies employing the six identical Web Services provided by Amazon in different countries.
3) Determine the best number of replicas for a selected replication strategy.

4. Result and Discussion

4.1. Reliability Assessment of Web Services

Tables 1-5 show the assessment results of the target Web Services. In the first colum of these tables, *a-us, a-jp, a-de, a-ca, a-fr* and *a-uk* stand for the six identical Amazon Web Services located in US, Japan, Germany, Canada, France, and UK, respectively. *GW* and *GIP* stand for the corresponding Global Weather Web Service and GeoIP Web Service. In the first row of the tables, *Cases* shows the failure rate (*R%*), which is the number of failed test cases (*Fail*) divided by the number of all executed test cases (*All*). *RTT* shows the average (*Avg*), standard deviation (*Std*), minimum (*Min*) and maximum (*Max*) values of the testing communication round-trip-times. *ProT* shows the average (*Avg)* and standard deviation (*Std*) values of process times of test cases in service servers. The *ProT* of *GW* and *GIP* are unavailable, as these two Web Services do not provide the information. Only values of correct cases are calculated in the *RTT* and *ProT*, because most of the failed cases have large *RTT* values and unavailable *ProT* values, which will affect accuracy of the result. All time units are in milliseconds (ms).

Table 1. Availability and RTT (Hong Kong)

	Cases			RTT(ms)				ProT	
	R%	Fail	All	Avg	Std	Min	Max	Avg	Std
a-us	0.32	110	34395	453	309	250	9562	42	21
a-jp	0.13	45	34425	407	340	203	9937	44	33
a-de	2.25	761	33889	587	348	343	9750	43	18
a-ca	0.45	153	34361	484	325	250	9515	43	20
a-fr	2.32	784	33771	588	358	343	9750	44	20
a-uk	2.51	844	33660	619	375	328	9813	43	20
GW	4.27	1461	34239	1582	1508	406	9989	-------	
GIP	3.75	1285	34284	847	1470	203	9984	-------	

Table 1 shows assessment results of the eight target Web Services in the locations of Hong Kong. Failure rate and RTT performance of *a-jp* are better than others. This means Hong Kong users can employ *a-jp* to achieve higher availability and smaller network latency. RTT standard deviation values of target Web Services are all very large, which indicates RTT values are changing drastically from time to time (For example, RTT values of *a-jp* are fluctuating from 203 millisecond to 9937 milliseconds in our experiment).

Failure rates and RTT performance of *GW* and *GIP* are not so good comparing with the six Amazon Web Services. However, this is understandable, because they are neither commercial nor designed for critical purposes. *ProT* values of Amazon Web Services are rather small comparing with RTT, indicating that latency of Amazon Web Services is mainly caused by Internet connection instead of Web Server executions. Among all the 5443 failure cases, 2986 failure cases are due to timeout (larger than 10 seconds), 2456 failure cases are due to unavailable service (http code 503), and 1 failure case is due to bad gateway (http code 502).

Table 2. Availability and RTT (Mainland China)

	Cases			RTT(ms)				ProT	
	R%	Fail	All	Avg	Std	Min	Max	Avg	Std
a-us	7.37	109	1479	1801	2048	484	9906	42	19
a-jp	7.52	128	1703	1683	1945	296	9937	46	27
a-de	7.41	114	1539	1806	1909	390	9844	42	17
a-ca	6.44	111	1723	1671	1961	281	9953	45	21
a-fr	6.41	96	1498	1869	1920	360	9999	43	18
a-uk	5.36	100	1865	1792	1903	359	9875	45	20
GW	26.3	337	1280	3487	2416	953	9969	-------	
GIP	1.66	32	1923	1230	1535	250	9781	-------	

Table 3. Availability and RTT (Australia)

	Cases			RTT(ms)				ProT	
	R%	Fail	All	Avg	Std	Min	Max	Avg	Std
a-us	0	0	1982	1218	1543	375	9937	42	16
a-jp	0	0	1996	1052	1465	312	9922	44	29
a-de	0	0	1935	1476	1603	453	9891	45	115
a-ca	0	0	1982	1190	1487	375	9828	42	17
a-fr	0	0	1823	1309	1350	453	9812	44	23
a-uk	0.15	3	1984	1326	1388	453	9734	44	24
GW	0.28	5	1794	1466	1590	234	9812	-------	
GIP	0	0	1994	875	1619	234	9899	-------	

Table 4. Availability and RTT (Taiwan)

	Cases			RTT(ms)				ProT	
	R%	Fail	All	Avg	Std	Min	Max	Avg	Std
a-us	0	0	9316	712	293	234	9422	44	22
a-jp	0.01	1	9328	596	276	187	5016	44	40
a-de	0	0	9895	938	369	343	9515	44	17
a-ca	0.06	5	8861	715	306	235	9791	43	20
a-fr	0	0	9537	923	388	329	9719	44	21
a-uk	0.02	2	8831	944	405	328	9762	45	23
GW	0.37	35	9344	1824	1445	250	9948	-------	
GIP	0.68	60	8774	793	1262	234	9922	-------	

Table 5. Availability and RTT (USA)

	Cases			RTT(ms)				ProT	
	R%	Fail	All	Avg	Std	Min	Max	Avg	Std
a-us	0	0	8004	100	135	31	5094	42	18
a-jp	0	0	8087	302	247	109	9282	43	33
a-de	0	0	7753	313	229	109	9390	43	16
a-ca	0	0	7794	177	217	31	9515	43	19
a-fr	0	0	8168	383	194	125	3906	44	30
a-uk	0	0	8060	318	228	124	9453	44	19
GW	0	0	7694	1189	1376	62	9855	-------	
GIP	0	0	7733	660	1354	62	9938	-------	

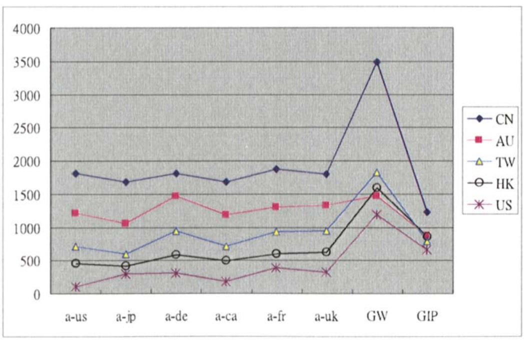

Figure 4. Avg RTT from Different Locations

Tables 2-5 are assessment results from user locations in Mainland China, Australia, Taiwan and USA. As shown in Figure 4, the average RTT values are quite different in these locations, where USA is the best and Mainland China the worst. Moreover, the failure rate in USA is the lowest while in Mainland China is the highest. It can be observed that the unstable Internet environment is the main contribution to unavailability of the target Web Services. Note the execution times for the Amazon Web Service, as shown by *ProT*, are essentially the same for all geographic sites. Even from the locations in USA, which enjoy the best RTT performance, the standard deviation of RTT is still very large. This large fluctuation of RTT can break down latency-sensitive applications. Approaches are needed to obtain more stable and reliable services performance.

4.2. Assessment of Replication Strategies

To enhance the service reliability as well as performance, WS-DREAM can be used to conduct assessment of various replication strategies. To clearly show the performance of these strategies in erroneous networking conditions, fault injection techniques [7,8] are applied to generate faulty test cases. Table 6 shows the assessment result of the correct cases and the faulty cases under various strategies. Tolerated failures are not shown in Table 6 although we have record on them, since they are transparent to the user.

Table 6. Performance of Replication Strategies

	Correct Cases					Faulty Cases				
	R%	Fail	All	Avg	Std	R%	Fail	All	Avg	Std
1	0.03	11	33466	273	160	0.57	19	3314	264	132
2	0.01	3	34545	378	333	0.03	1	3709	720	705
3	0	0	34293	367	295	0.03	1	3676	810	721
4	0.02	6	34003	466	388	0.28	10	3621	529	564
5	0	1	33104	468	384	0.14	5	3569	767	515
6	0.01	2	32955	463	359	0.06	2	3542	528	446
7	0.01	2	32806	464	393	0.03	1	3552	790	476
8	0	0	33076	404	1160	0	0	3638	729	863
9	0	0	33066	378	299	0	0	3514	731	674

As shown in Table 6, there are 25 failures in the normal environment (*Correct Cases*), and 39 failure cases in the heavy faulty environment (*Faulty Cases*). All failure cases are due to timeout of all six replicas, which may be caused by the client-side network problems. Most of these failure cases occur in Strategy 1 (*Parallel*), which may be caused by too many simultaneous network connections.

In both normal and faulty environment, Strategy 1 (*Parallel*) provides the best RTT performance. This is reasonable, for it used the first properly returned response as the final outcome. The sequential-type strategies (i.e., 2:Retry, 3:RB, 8:Retry+RB, and

9:RB+Retry) can provide good RTT performance in the normal environment, since in most cases they only need to run one replica to get the correct result. However, in the faulty environment, their performances are not so good; because they need to try another backup replica or themselves sequentially when fail.

We can observe that the most suitable strategy is application dependent. For example, real-time applications can employ Strategy 1 to obtain better RTT performance, while payment oriented applications can use the sequential-type strategies, since parallel strategies will lead to multi-payments.

4.3. Assessment of Best Replica Number

From the experimental data, Ben decides to use the Strategies 1 (*Parallel*) to obtain better latency and reliability performance of his commercial Web site. The next question to ask is: How many replicas should be used in the Strategy 1? WS-DREAM can be employed again to make assessment. The result is shown in Table 7. To show the performance with different number of replica, fault test cases are applied to simulate 5% faulty Internet environment.

Table 7. Performance of Replica Numbers

	Correct Cases					5% Fault				
	R%	Fail	All	Avg	Std	R%	Fail	All	Avg	Std
2	0.01	1	19375	324	257	0.77	140	18281	331	280
3	0.04	6	14857	299	192	0.01	2	14169	300	203
4	0.01	1	14317	283	162	0.02	3	14305	287	195
5	0.06	8	14226	272	167	0.04	6	14180	276	156
6	0.03	4	14100	273	168	0.01	1	14163	270	177

As shown in Table 7, two replicas are enough to provide high availability in the normal Internet environment, while three replicas are needed to ensure high reliability in the 5% faulty Internet environment. No matter how many replicas used in the Strategy 1, a few failure cases still occur due to time out of all replicas, which may cause by client side network problems. The overall failure rate performance with Strategy 1 is much better than without any replication strategies, since a lot of logical faults and communication faults have been masked by the replication strategy. In our experiment, Strategy 1 (*Parallel)* with six replicas obtains the best RTT performance, but not significant.

Based on the experimental result, Ben decides to employ Strategy 1 (*Parallel)* with 3 replica (e.g., jp, ca, us) to build a reliable Service Oriented Web site.

5. Conclusion

This paper presents a practical distributed mechanism named WS-DREAM for assessing reliability of Web services based on user collaboration schemes in real-world applications. The experimental results indicate that unstable Internet environments and server connections can lead to unreliability of Web services. With the facility of WS-DREAM, accurate reliability assessment of target Web services can be automatically acquired in a user collaboration manner, and optimal replication strategies engaging fault tolerance and design diversity schemes can be effectively obtained for the achievement of Web Service reliability.

Our future work includes an automatic mechanism for users to search for similar or identical Web Service as replica for design diversity purpose, and the enhancement of system feature in facilitating user test case contributions.

6. Acknowledgement

The work described in this paper was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK4150/07E), and a grant from the Research Committee of The Chinese University of Hong Kong (Project No. CUHK3/06C-SF).

7. References

[1] X. Bai, W. Dong, W.T. Tsai and Y. Chen (2005), "WSDL-Based Automatic Test Case Generation for Web Services Testing", Proceedings of IEEE Workshop on Service-Oriented System Engineering, pp. 207 – 212, 2005.
[2] H. Foster, S. Uchitel, J. Magee, and J. Kramer (2003), "Model-based Verification of Web Service Compositions", Proceedings of 18th IEEE International Conference on Automated Software Engineering, pp.152-161, 2003.
[3] P. W. Chan, M.R. Lyu and M. Malek, "Reliable Web Services: Methodology, Experiment and Modeling," in IEEE International Conference on Web Services, 2007.
[4] N. Salatge and J.C. Fabre, "Fault Tolerance Connectors for Unreliable Web Services," in 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), Edinburgh, UK, pp. 51-60, 2007.
[5] Wikipedia, http://www.wikipedia.org
[6] BitTorrent, http://www.bittorrent.com
[7] N. Looker and J. Xu, "Assessing the Dependability of Soaprpc-based Web Services by Fault Injection," In Proc. of the 9th IEEE International Workshop on Object-oriented Real-time Dependable Systems, pp. 163-170, 2003.
[8] M. Vieira, N. Laranjeiro, and He. Madeira, "Assessing Robustness of Web-Services Infrastructures," in 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), Edinburgh, UK, pp. 131-136, 2007.
[9] M.R. Lyu, "Software Fault Tolerance", Wiley Trends in Software book series, John Wiley & Sons, Chichester, February 1995.
[10] WS-DREAM Web site, http://137.189.89.120.