Towards ServMark
An Architecture for Testing Grids

M. Andreica and N. Tăpus
Polytechnic University of Bucharest and
A. Iosup * and D.H.J. Epema *
Delft Technical University and
C.L. Dumitrescu *
University of Münster and
I. Raicu and I. Foster
The University of Chicago and
M. Ripeanu
The University of British Columbia

* This research work is carried out under the FP6 Network of Excellence CoreGRID
funded by the European Commission (Contract IST-2002-004265) and is available as
CoreGRID TechReport #00062.
INTRODUCTION

WHY GRID SERVICE BENCHMARKING?

→ Production Grids range from a few hundred to several tens of thousands of resources and services, and from few to thousands of users

→ There is a need to understand and quantify the behavior and performance of such environments

→ Evaluating the performance of services provided for a large number of wide-area distributed users is a non-trivial endeavor
Talk Outline

→ Introduction

→ Requirements for Grid Testing

→ ServMark - A Grid Performance Testing Framework and the Starting Points

→ DiPerF, GrenchMark, and ServMark Design Presentations

→ Performance Metrics

→ ServMark Validation, Setup, and Results

→ Conclusions, Future Work, and Acknowledgments
Requirements for Grid Testing

1. **Representative workload generation:** testing tools must create the conditions that the Grids were designed for.

2. **Accurate testing:** collected performance metrics are heavily dependent on the accuracy of various mechanisms employed by the testing framework.

3. **Scalable testing:** the testing framework must be at least as scalable as tested system.

4. **Reliable testing:** the testing framework must detect and account for its own failures, especially when operating in wide-area environments.

5. **Extensibility, Automation, and Ease-of-Use:**
   - usability of a testing system is at least as important as its features.
   - without extensibility support, already obtained results would become obsolete, as they would not be comparable with those for the new systems.
   - automation and the ease-of-use can be summarized as *single-click testing procedure*.
SERVMARK
A GRID PERFORMANCE TESTING FRAMEWORK

→ Is aimed at simplifying and automating the testing process in real Grid environments

→ Is capable of:
  • coordinating a pool of machines that test a target service
  • generating complex testing workloads
  • collecting and aggregating performance metrics
  • generating performance statistics

→ Aggregates data and provides information on:
  • service throughput
  • service fairness when serving multiple clients concurrently
  • the impact of network latency on service performance, effectively enabling functionality and scalability testing
A GLOBUS INCUBATOR PROTOPROJECT

ServMark: A Distributed Grid and Services Testing Framework

- Incubator ProtoProject
- Allows
  - Testing different parts of the Globus Toolkit™ v.3/v.4, e.g., MDS, GridFTP, pre-WS GRAM and WS GRAM;
  - Testing new (Globus) services developed by end-users, e.g., DI-GRUBER, Koala;
  - Functionality testing and system tuning, e.g., Koala, The Distributed ASCI Supercomputer (DAS).
- Successfully deployed in
  - Environments where the Globus Toolkit™ is deployed, e.g., Grid3, TeraGrid, DAS, CS@UChicago cluster;
  - Other large-scale environments, e.g., PlanetLab.

http://dev.globus.org/wiki/Incubator/ServMark

ServMark Importance

- Test Suite for Services deployed in large-scale environments (Grids)
- Blends DIPerF and GrenchMark, two Grid performance evaluation tools.
- Tackles two orthogonal issues in Grid performance evaluation:
  - Multi-sourced testing (multi-user scenarios, scalability)
  - Generate and run dynamic test workloads with complex structures (real-world scenarios, flexibility)

http://dev.globus.org/wiki/Incubator/ServMark

- Accepted: June 11, 2006
- Re-evaluated every quarter until either escalates into a full project or becomes obsolete
- Currently there are six comitters, but contributions and new members are welcome
- Presented at GGF’06 and SC’06 as part of the Globus Incubator (Proto)Projects
DiPerF coordinates a pool of machines, collects and aggregates metrics, and generates statistics.
→ **GrenchMark** generates and submits complex workloads
DiPerF Design
Four Components

- **The analyzer** receives the performance metrics from the distribution system and perform different operation on them in order to compute, for example, throughput.

- **The controller** coordinates the performance evaluation experiment; it distributes the client code to testers via submitters and coordinates testers’ activity.

- **Submitters** provide the interface for distributing the testers and the testing client to the underlying testing environment (e.g., a Grid, a cluster resource manager, a P2P system, etc.).

- **Each tester** runs the client code on its local machine and times their (RPC like) access to the target service.
GRENCHMARK DESIGN

→ Framework for synthetic Grid workload generation and submission developed in the Distributed Systems Group at Technical University of Delft, the Netherlands

→ Allows new types of Grid applications to be included in the workload generation, or user’s parameterization of the generation and submission processes

→ Based on the concepts of unit generators, of job description files (JDF) printers, and of printers:
  - job description files describe at a high-level user’s desired workloads
  - unit generators produce detailed descriptions on running a set of applications (workload unit) based on the description files
  - printers take the generated workload units and create end-system files suitable for grid submission

→ Multiple unit generators can be coupled to produce a workload that can be submitted to a resource manager
Refined Requirements for ServMark

1. Uniquely identify each test (*REQ1*)
2. Automatically generate a multi-node test according to the user specifications (*REQ2*)
3. Store the test and make it available for replay (*REQ3*)
4. Run the test and store its results permanently (*REQ4*)
5. Analyze the results and compute complex statistics (*REQ5*)
6. Performance evaluation must be online: results should be able to be visualized as the testing process advances (*REQ6*)
ServMark blends DiPerF and GrenchMark, and:

- adds the necessary coordination and automation layer
- testers’ management becomes user-transparent
- complex workloads are described by a few characteristics
ServMark’s Default Performance Metrics

→ **Service processing time**: time from request issuance to reply receiving

→ **Service throughput**: number of requests completed averaged over a short time interval

→ **Offered load**: number of concurrent service requests

→ **Service fairness**: standard deviation for utilization when all clients are active

→ **Job success rate (per client)**: the ratio of jobs that were successfully completed for a particular client

→ **Job fail rate (per client)**: the ratio of jobs that failed for a particular client

→ **Time to job completion (TTJC)**: difference between the moment of successful completion and the previous moment of a successful completion

→ **Time to job failure (TTJF)**: for every failed job, the difference between the moment of failure and the previous moment of a failure
MORE PERFORMANCE METRICS
INTERNAL AND USER-DEFINED

→ **Ping time**: the time required for a UDP packet round-trip

→ **Clock skew among tester nodes**: the time difference among the controller and tester node, and used for clock fixing

→ **User-defined**: any metric reported by a client under the following format:

   name: time (10^{11}.000000) value (0.00000) host.00000
Towards Real Grid Testing, ServMark ... 

→ makes use of the properties of both its constituent systems in order to generate truly significant testing scenarios. First, by using a distributed approach.

→ is able to generate a wide range of testing conditions for many Grid environments and services.

→ synchronizes the time between client nodes with a synchronization error smaller than 100ms.

→ detects client failures during the test, and reports the failure impact on the obtained results’ accuracy.

→ can be automated to the degree of a single-click testing procedure, especially for periodic functionality or performance testing repository.
SERVMARK VALIDATION
BY MEANS OF …

→ tests on fine-grained services deployed in real large-scale environments, which is a difficult aspect of the generic problem of testing services in P2P and Grid environments

→ machines from PlanetLab (a World-scale System) and DAS-2 (the Dutch Grid System) testbeds

→ measurements on the performance of six most-used web servers: Apache, Null HTTPD, Apache Tomcat, Nweb, Jetty and Awhttpd

→ results capturing:
  • services’ maximum throughput
  • service “fairness” when multiple clients access the service concurrently
  • the impact of network latency on service performance from both client and service viewpoint
Testing Setup

- **ServMark controller** was installed on s8.uchicago.edu located at the University of Chicago

- **Web servers** were started on alice01.rogrid.pub.ro located at the Polytechnic University of Bucharest

- **ServMark testers** were spawned on machines part of PlanetLab
  
  - for each test, 20 testers were run on hosts from North and South America, Asia, and Europe
  
  - each ServMark tester launched 100 HTTP requests, with a Poisson inter-arrival time distribution of $\lambda = 1s$
  
  - a request which remained unanswered for more than 25 seconds was considered to be faulty and was terminated
## Results for Comparing Six Web Servers in WAN

Table 1: Service processing time for the six web servers (s)

<table>
<thead>
<tr>
<th>Web Server</th>
<th>Average (Std. Dev.)</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Weighted Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>1.0779 (0.647)</td>
<td>0.0810</td>
<td>16.5440</td>
<td>1.0969</td>
</tr>
<tr>
<td>Null HTTPD</td>
<td>0.9442 (0.482)</td>
<td>0.1244</td>
<td>30.4872</td>
<td>0.9495</td>
</tr>
<tr>
<td>Apache Tomcat</td>
<td>1.3617 (0.732)</td>
<td>0.1724</td>
<td>24.2665</td>
<td>1.3930</td>
</tr>
<tr>
<td>Nweb</td>
<td>0.9731 (0.565)</td>
<td>0.1293</td>
<td>10.9908</td>
<td>1.0152</td>
</tr>
<tr>
<td>Jetty</td>
<td>10.0745 (1.210)</td>
<td>0.2651</td>
<td>35.4375</td>
<td>9.0297</td>
</tr>
<tr>
<td>Awhttpd</td>
<td>1.1739 (0.558)</td>
<td>0.1242</td>
<td>29.5580</td>
<td>1.0117</td>
</tr>
</tbody>
</table>
Table 2: Time to job completion (TTJC) for the six servers (s)

<table>
<thead>
<tr>
<th>Web Server</th>
<th>Average (Std. Dev.)</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Weighted Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>3.8803 (1.975)</td>
<td>0.0022</td>
<td>13.5419</td>
<td>3.6702</td>
</tr>
<tr>
<td>Null HTTPD</td>
<td>3.9409 (1.922)</td>
<td>0.0177</td>
<td>11.7235</td>
<td>3.7446</td>
</tr>
<tr>
<td>Apache Tomcat</td>
<td>4.0902 (2.061)</td>
<td>0.0034</td>
<td>13.8347</td>
<td>3.8399</td>
</tr>
<tr>
<td>Nweb</td>
<td>4.0870 (2.008)</td>
<td>0.0393</td>
<td>14.1707</td>
<td>3.8613</td>
</tr>
<tr>
<td>Jetty</td>
<td>6.4677 (1.582)</td>
<td>0.0010</td>
<td>15.0310</td>
<td>5.9648</td>
</tr>
<tr>
<td>Awhttpd</td>
<td>4.1798 (2.041)</td>
<td>0.0106</td>
<td>13.9180</td>
<td>3.9005</td>
</tr>
</tbody>
</table>
Table 3: Time to job failure (TTJF) for the six web servers (s)

<table>
<thead>
<tr>
<th>Web Server</th>
<th>Average (Std. Dev.)</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Weighted Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>No Failures</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Null HTTPD</td>
<td>2.7893 (0.000)</td>
<td>0.0000</td>
<td>5.5786</td>
<td>2.7893</td>
</tr>
<tr>
<td>Apache Tomcat</td>
<td>No Failures</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Nweb</td>
<td>No Failures</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Jetty</td>
<td>1.4840 (0.000)</td>
<td>0.000</td>
<td>17.8760</td>
<td>1.4840</td>
</tr>
<tr>
<td>Awhttpd</td>
<td>No Failures</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Results show the existence of three classes of web servers, very fast, fast and slow:

- **very fast class**: Nweb, Null HTTPD, and Apache
- **fast class**: Apache Tomcat, and Awhttpd
- **slow class**: Jetty web server

Observations:

- very large service processing times in the case of the Jetty web server
- Jetty web server is the only one using the Java platform and the Java Virtual Machine used during our tests was non-commercial
CONCLUSIONS

- Initial results demonstrate the operation of ServMark when testing fine-grained services in large-scale environments.

- ServMark was tested on DAS-2 for simple shell commands, and PlanetLab for comparing the performance of six Web servers.

- ServMark fulfills the main requirements for testing in large environments: generates workloads, provides accurate testing, is scalable and reliable, and supports extension.

- In practice, ServMark can be easily used for complete automated testing (one-click scalability testing).

- We conclude that ServMark is useful for testing P2P and Grid services and environments.
**Future Work**

- Better interface between users and the ServMark controller for alternative testing scenarios

- Alternative ways to send information from testers to the controller, i.e., through configurable push/pull mechanisms

- Integration with other systems for distributed resource management (e.g., Condor, gLite)

- Complete porting to a fault-tolerant Grid service

- Many more tests and scenarios for validation purposes (ranging from monitoring systems, such as MonAlisa, to P2P systems, such as Tribbler)
THANKS …

ServMark is available from the Globus Incubator Project Web Site: http://dev.globus.org/wiki/Incubator/ServMark

This TechReport (#0062) is available at:
http://www.coregrid.net/mambo/content/view/209/203/

Acknowledgments:

- Part of this research work was supported by the University of Chicago
- This research work is also carried out partially under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265)
- Part of this work was also carried out in the context of the Virtual Laboratory for e-Science project (http://www.vl-e.nl), which is supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science (OC&W), and which is part of the ICT innovation program of the Dutch Ministry of Economic Affairs (EZ)
- The integration work was supported mainly by the EU-NCIT - NCIT leading to EU IST Excellency project, EU FP6-2004-ACC-SSA-2
And

... Any Questions?