European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
A Network of Excellence funded by the European Commission

Towards ServMark, an Architecture for Testing Grids

M. Andreica, N. Tapus
Computer Science Department
Polytechnic University of Bucharest
e-mail: mugurel,tapus@cs.pub.ro

A. Iosup, D.H.J. Epema
Electrical Eng., Mathematics and Computer Science Department
Technical University of Delft
e-mail: A.Iosup,D.H.J.Epema@tudelft.nl

C. Dumitrescu
Department of Mathematics and Computer Science
The University of Münster
e-mail: dumitres@uni-muenster.de

I. Raicu, I. Foster
Computer Science Department
The University of Chicago
e-mail: iraicu,foster@cs.uchicago.edu

M. Ripeanu
Electrical and Computer Engineering
The University of British Columbia
e-mail: matei@ece.ubc.ca

CoreGRID Technical Report
Number TR-0062
November 28, 2006

Institute on Resource Management and Scheduling (WP 6)

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

Towards ServMark, an Architecture for Testing Grids

M. Andreica, N. Tapus
Computer Science Department
Polytechnic University of Bucharest
e-mail: mugurel,tapus@cs.pub.ro

A. Iosup, D.H.J. Epema
Electrical Eng., Mathematics and Computer Science Department
Technical University of Delft
e-mail: A.Iosup,D.H.J.Epema@tudelft.nl

C. Dumitrescu
Department of Mathematics and Computer Science
The University of Münster
e-mail: dumitres@uni-muenster.de

I. Raicu, I. Foster
Computer Science Department
The University of Chicago
e-mail: iraicu,foster@cs.uchicago.edu

M. Ripeanu
Electrical and Computer Engineering
The University of British Columbia
e-mail: matei@ece.ubc.ca

CoreGRID TR-0062

November 28, 2006

Abstract

The Grid promise is starting to materialize today: large-scale multi-site infrastructures have grown to assist the work of scientists from all around the world. In only ten years, production Grid environments have grown from a few hundred to several tens of thousands of resources, and from few to hundreds of users. To exploit this already existing infrastructure, the behavior of the real systems, and in particular their offered performance, must be understood and quantified. However, evaluating the performance testing in such large-scale environments is a non-trivial endeavor, for which no comprehensive solution exists. To address this problem, we present in this work our first steps towards ServMark, a performance testing framework aimed at simplifying and automating the testing process in Grid environments. ServMark coordinates a pool of machines that test a target service, generates complex testing workloads, collects and aggregates performance metrics, and generates performance statistics. The aggregate data collected provide information on service throughput, on service fairness when serving multiple clients concurrently, and on the impact of network latency on service performance, effectively enabling functionality and scalability testing. Our initial results demonstrate the operation of ServMark when testing fine-grained services in real environments.

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

1 Introduction

The Grid world is starting to fulfill the promise of a world-scale computing infrastructure for the use of the ever-growing scientific community. In a decade, systems comprising a few hundreds of resources have grown tremendously, with current systems such as CERN's LCG, NorduGrid, TeraGrid, Grid'5000, or The Open Science Grid, gathering together (tens of) thousands of resources, and offering similar or better throughput when compared with large-scale parallel production environments [41,42]. To exploit this already existing infrastructure, the behavior of the systems, and in particular the offered performance, must be understood and quantified. Using performance evaluation data it is possible to build empirical performance estimators that link observed service performance (throughput, response time) to offered load. These estimates can be then used as input by a resource scheduler to increase resource utilization while maintaining desired quality of service levels. We argue that only real environment testing can provide accurate performance insights. Indeed, the Grid systems complexity renders theoretical performance evaluations unpractical, and the dynamicity, the heterogeneity, or even the sheer size of the current grid systems make simulation results applicable only for first-order evaluations. However, evaluating the performance testing in real large-scale environments is a non-trivial endeavor, for which no comprehensive solution exists.

To address this problem, we present in this work our first steps towards ServMark, a performance testing framework aimed at simplifying and automating the testing process in real Grid environments. ServMark is based on our previous work on DiPerF [17] and GrenchMark [18]. DiPerF [17] is built around the idea of coordinating a distributed pool of machines that run clients of a target service, collects and aggregates performance metrics, and generates performance statistics. GrenchMark [18] focuses on generating complex workloads for testing purposes. ServMark couples these two approaches, and adds the necessary coordination and automation layer. Thus, ServMark coordinates a pool of machines that test a target service, generates complex testing workloads, collects and aggregates performance metrics, and generates performance statistics. The aggregate data collected provide information on service throughput, on service fairness when serving multiple clients concurrently, and on the impact of network latency on service performance, effectively enabling functionality and scalability testing. Our initial results demonstrate the operation of ServMark when testing fine-grained services deployed in real large-scale environments. Using machines from the PlanetLab [46] and the Grid3 [47] testbeds, we conduct experiments in which the service clients experience different levels of connectivity. The data collected provide information on services maximum throughput, on service fairness when multiple clients access the service concurrently, and on the impact of network latency on service performance from both client and service viewpoint. We conclude that ServMark is useful for testing P2P and Grid ideas in real large-scale systems.

The remainder of this paper is structured as follows. The following section presents our motivation. Section 3 describes the ServMark frameworks design. The validation of the framework is presented in Section 4. The paper concludes with a brief summary of our experience and future work plans.

2 Motivation and Goals

Grid computing [23] provides a natural way to aggregate resources from different administrative domains for building large scale distributed environments [2]. The Web Services paradigm [24] proposes a way by which virtual services can be seamlessly integrated into global-scale solutions to complex problems. While the usage of Grid technology ranges from academia and research to business world and production, two issues must be considered: that the promised functionality can be accurately quantified and that the performance can be evaluated based on well defined means. Without adequate functionality demonstrators, systems cannot be tuned or adequately configured, and Web services cannot be stressed adequately in production environment. Without performance evaluation systems, the system design and procurement processes are limp, and the performance of Web Services in production cannot be assessed.

In Section 1 we have argued for the importance of performance testing in real environments. We further detail the main requirements for testing in real grid environments:

- *Representative workload generation.* In order for the results to be significant, the testing tool must be able to create the conditions that the Grid environments (or their components) were designed to handle [4, 12, 46, 47]. Consider the case of a resource management system. Here, the system users submit jobs according to daily patterns [9, 15, 40], and may respond to the systems feedback, i.e., they will not continue to submit until their already submitted jobs are finished [41]. It would therefore be interesting to establish the performance of the resource management system under both real-life and extreme conditions.

- *Accurate testing.* The accuracy of the performance metrics collected is heavily dependent on the accuracy of the timing mechanisms used and on accurate time synchronization among the testing machines.
- *Scalable testing.* The scalability of the testing framework must be at least that of the scalability of tested system. Because the number of resources to be found in nowadays Grids is on the order of thousands to tens of thousands [42], and because the size is expected to grow, the evaluation system must generate and coordinate significant loads, in a scalable way.
- *Reliable testing.* The testing framework must detect and account for its own failures, especially when operating in wide-area environments.
- *Extensibility, Automation, and Ease-of-Use.* The usability of a testing system is at least as important as its features. We argue that it is the ease-of-use, the degree of automation, and the extensibility, that separate a successful tester from other similar approaches. The automation and the ease-of-use can be summarized as single-click testing procedure. Given the current evolution speed in the Grid world, that a testing system would become obsolete is only a matter of years. Without the ability to accommodate extensions, already obtained results would become obsolete, as they would not be comparable with those for the new systems.

3 The Design of ServMark

In this section we present the design of ServMark. which integrates two existing performance evaluation systems: DiPerF [17] and GrenchMark [18]. In addition to its components capabilities, ServMark adds the needed coordination and automation layer, for improved automation and ease-of-use (see Figure 1).

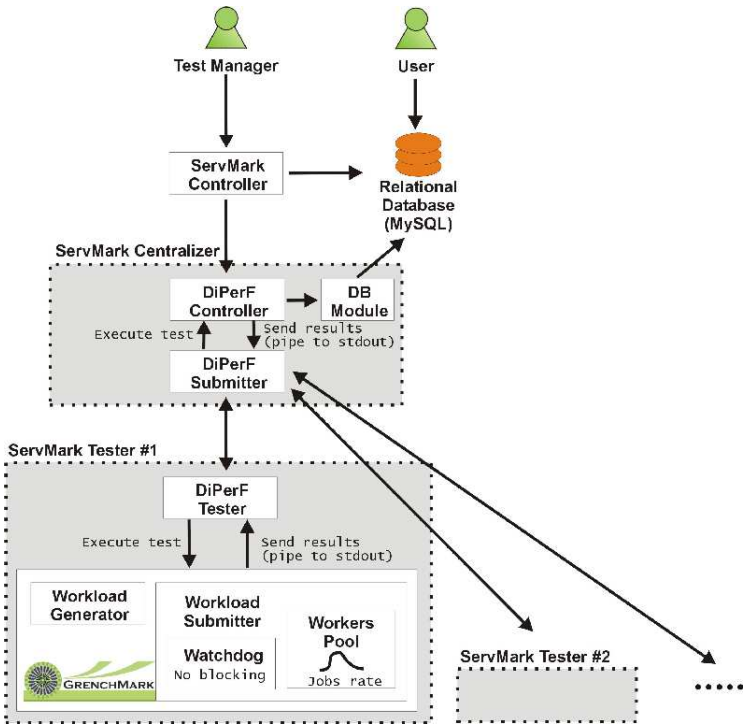

Figure 1: The ServMark Architecture.

3.1 The ServMark Components

ServMark is based on two components: DiPerF and GrenchMark. DiPerF is a distributed testing system and test generator, and GrenchMark is a centralized system that can generate complex testing scenarios. ServMark makes use

of the properties of both systems in order to generate truly significant testing scenarios.

DiPerF aims to simplify and automate service performance evaluation. DiPerF coordinates a pool of machines that access a centralized or distributed target service and collect performance metrics. Centralized DiPerF components then, aggregate these performance measurements and generate performance statistics. The aggregate data collected provides information on service throughput, service response time, service fairness when serving multiple clients concurrently, and on the impact of network latency on service performance. All steps involved in this process are automated.

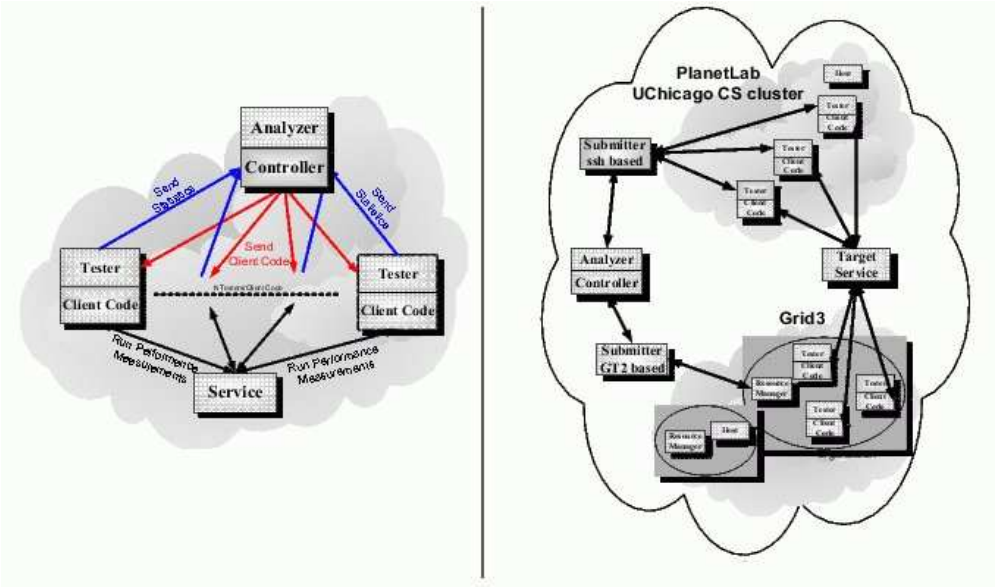

Figure 2: DiPerF Overview and Deployment Scenario Exemplification

DiPerF consists of four major components: the analyzer, the controller, the submitters and the testers. A user of the framework provides to the controller the location of the target service to be evaluated and the client code to access the service. The controller then coordinates the performance evaluation experiment: it distributes the client code to testers via submitters and coordinates testers activity. Each tester runs the client code on its local machine and times their (RPClike) access to the target service. Finally, the controller collects all the measurement data from testers and performs additional operations (e.g., reconciles time stamps from various testers) to compute aggregated performance views. Sophisticated clients can have complex interactions with the target service and return periodic feedback and user defined metrics to the tester be propagated back to the controller.

GrenchMark is a framework for synthetic Grid workload generation and submission, which has been designed, implemented, and deployed by the MultiProbe team in the Parallel and Distributed Systems group of the Faculty of Electrical Engineering, Mathematics, and Computer Science of the Delft University of Technology. GrenchMark is extensible, in that it allows new types of grid applications to be included in the workload generation, parameterizable, as it allows the user to parameterize the workloads generation and submission, and portable, as its reference implementation is written in Python. The workload generator is based on the concepts of unit generators and of job description files (JDF) printers. The unit generators produce detailed descriptions on running a set of applications (workload unit), according to the workload description provided by the user. In principle, there is one unit for each supported application type. The printers take the generated workload units and create job description files suitable for grid submission. In this way, multiple unit generators can be coupled to produce a workload that can be submitted to any grid resource manager, as long as the resource manager supports that type of applications. Currently, GrenchMark can submit jobs to the KOALA, Globus GRAM, and Condor resource management systems.

GrenchMark offers support for the following workload modeling aspects. First, it supports unitary (e.g., sequential, parallel jobs using MPI, malleable/evolving jobs using Java/Ibis) and composite applications (e.g., workflows in the form of Directed Cyclic Graphs), single-site and co-allocated jobs. Second, it allows the user to define various job inter-arrival times based on well-known statistical distributions. Besides the Poisson distribution, used traditionally in queue-based systems simulation, GrenchMark also supports uniform, normal, exponential and hyper-exponential, Weibull, log normal, and gamma distributions. Third, it allows the workload designer to combine several workloads

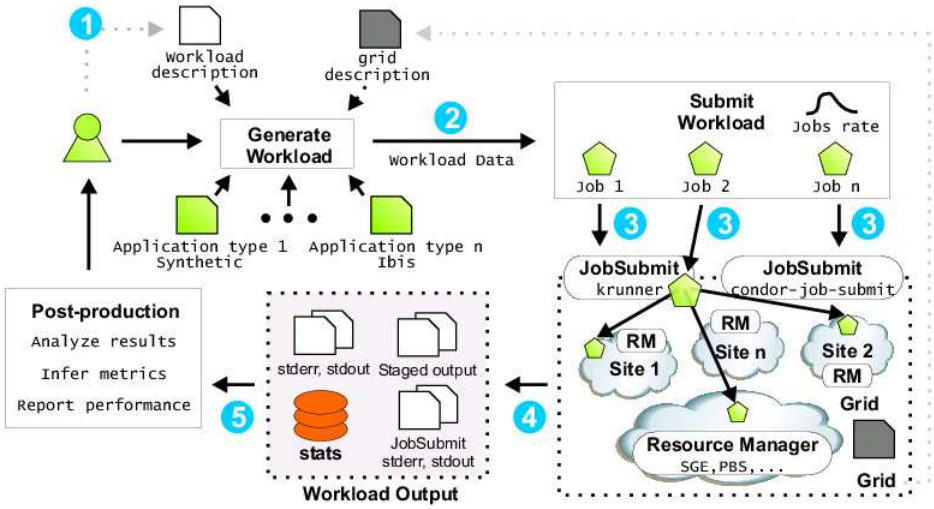

Figure 3: The GrenchMark Process

into a single one (mixing). This allows for instance the inclusion of bursts, by combining a short workload with many jobs per time unit with a longer one, comprising fewer jobs per time unit. An additional use of workload mixing is in a what-if analysis that evaluates what will happen to a grid community if its resources would be shared with another group of users. In this case, the workload modeler can mix the typical workload of the two communities and evaluate whether the system can support both, under various job acceptance and execution policies.

3.2 The Coordination and Automation Layer

The intended use for ServMark is to evaluate the performance of Grid environments and Grid and Web services. Grid environments and Web services have quite different behaviors in terms of response time, so different testing strategies need to be used. For ServMark, the testing process is initiated by a central controller, which distributes the testing parameters to multiple nodes. Each node generates its own test scenario based on the given parameters and then plays the generated scenario. The practical requirements are:

- uniquely identify each test (REQ1);
- automatically generate a multi-node test according to the user specifications (REQ2);
- store the test and make it available for replay (REQ3);
- run the test and store its results (REQ4);
- analyze the results and compute statistics (REQ5);
- the performance evaluation must be online: results should be able to be visualized as the testing process advances (REQ6).

Figure 1 shows the proposed architecture for ServMark, highlighting the relationship between GrenchMark, DiPerF and the new ServMark modules. The interaction between the user and the ServMark Controller goes as follows: the user decides the parameters to be used in the testing process (see REQ2), starts the ServMark Controller, and then is notified when the testing operation has completed. The ServMark Controller should generate a test ID for the testing process initiated by the user (see REQ1), update the database and send the testing parameters to the DiPerF controller. The DiPerF controller controls the testing process, by invoking the DiPerF submitter. It also updates the results into the database. The DiPerF submitter creates the tester processes and communicates with them, sending in parameters and receiving back test results. The DiPerF tester invokes GrenchMark, which performs the actual testing process and communicates with GrenchMark, sending parameters and receiving back test results. GrenchMark generates a workload according to the user parameters and then submits the generated workload for execution, computing the test

results and sending them to the DiPerF tester. The test parameters are inserted into the database by the ServMark controller. The DiPerF controller inserts and updates the test results into the database as the testing process advances.

3.3 The Performance Metrics

We have focused on flexibility in handling large data analysis tasks completely unsupervised. The performance analyzer is designed to allow a reduction of the raw performance data to a summary of the performance data with samples computed at a specified time quantum. For example, a particular experiment can accumulate more than one million performance samples over a period of and hour, but after the performance analyzer summarizes the data for one sample per second, the end result can be reduced to less than ten samples.

We also introduce the performance metrics considered by ServMark. While the performance metrics of interest to the user may vary from case to case, and our system allows the introduction and processing of user specified metrics, providing the following minimal set of pre-configured metrics [17,18]:

- *service processing time*: the time from when a client issues a request to the moment a reply is received minus the round-trip time to the service and minus the execution start-up time of the client code. This metric is measured from the point of view of the client;
- *service throughput*: number of requests completed successfully by the service averaged over a short time interval specified by the user (e.g., a second or a minute) in order to reduce the large number of samples. To make the results easier to understand most of the graphs below also present moving averages;
- *offered load*: number of concurrent service requests (per second); service utilization (per client): ratio between the number of requests completed for a specific client and the total number of requests completed by the whole service during the time the client was active;
- *service fairness*: the standard deviation in service utilization measured when all clients are active concurrently;
- *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)*: for every correctly completed job, the difference between the moment of successful completion and the previous moment of a successful job completion, or the beginning of the testing interval;
- *time to job failure (TTJF)*: for every failed job, the difference between the moment of failure and the previous moment of a failure, or the beginning of the testing interval.

4 Towards Real Grid Testing

While we acknowledge that a lot of ground must still be covered to fulfill the requirements of a system for testing Grid environments, and Web (and Grid) Services, we argue that ServMark addresses the main requirements for testing in real grid environments (see Section 2). 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. Second, by using a versatile workload generation engine, each testing unit of ServMark may generate complex workloads, both real (trace-based) and realistic (model-based);
- synchronizes the time between client nodes with a synchronization error smaller than 100ms. This ensures the accuracy of the testing procedure;
- 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. In particular, data collected by testers are automatically retrieved and stored in a central repository.

5 The ServMark Validation

In order to test the ServMark implementation, we chose to evaluate a scenario in which ServMark is used to test fine-grained services deployed in real large-scale environments, which we consider the most difficult aspect of the generic problem of testing P2P and Grid components in real large-scale systems. Using machines from the PlanetLab [46] and the Grid3 [47] testbeds, we conduct experiments in which the service clients experience different levels of connectivity. We test in this environment the performance of six of the most-used web servers: Apache, Null HTTPD, Apache Tomcat, Nweb, Jetty and Awhttpd. The data collected provide information on services maximum throughput, on service fairness when multiple clients access the service concurrently, and on the impact of network latency on service performance from both client and service viewpoint. While our results should not be used as indicators to what is the best web server (for this we should have devised much more realistic load, and should have used many more testing scenarios), we conclude that ServMark is useful for testing fine-grained services in real large-scale environments.

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

Web Server	Average (Standard Deviation)	Minimum	Maximum	Weighted Average
Apache	1.0779 (0.647)	0.0810	16.5440	1.0969
Null HTTPD	0.9442 (0.482)	0.1244	30.4872	0.9495
Apache Tomcat	1.3617 (0.732)	0.1724	24.2665	1.3930
Nweb	0.9731 (0.565)	0.1293	10.9908	1.0152
Jetty	10.0745 (1.210)	0.2651	35.4375	9.0297
Awhttpd	1.1739 (0.558)	0.1242	29.5580	1.0117

Table 2: TTJC for the six web servers (in seconds))

Web Server	Average (Standard Deviation)	Minimum	Maximum	Weighted Average
Apache	3.8803 (1.975)	0.0022	13.5419	3.6702
Null HTTPD	3.9409 (1.922)	0.0177	11.7235	3.7446
Apache Tomcat	4.0902 (2.061)	0.0034	13.8347	3.8399
Nweb	4.0870 (2.008)	0.0393	14.1707	3.8613
Jetty	6.4677 (1.582)	0.0010	15.0310	5.9648
Awhttpd	4.1798 (2.041)	0.0106	13.9180	3.9005

5.1 Experimental Setup

The ServMark controller was installed on *s8.diperf.cs.uchicago.edu*, a machine located at the University of Chicago, Computer Science Department. The web servers were started on *alice01.rogrid.pub.ro*, a machine located at the Polytechnic University of Bucharest. The testers were spawned on machines which are part of PlanetLab [46]. PlanetLab currently consists of over 600 machines hosted by over 300 sites, and is spanning over 25 countries. For each test, 20 testers were selected to run on hosts from North and South America, Asia, and Europe, simultaneously. Each ServMark tester was configured to launch 100 HTTP requests, with a Poisson inter-arrival time distribution of λ = 1s. A request which remained unanswered for more than 25 seconds was considered to be faulty and was, subsequently, killed.

5.2 Validation: Testing Fine-Grained Services

Table 3 presents the statistical values for the service processing time of the six web servers we tested. For the selected scenario, the results have shown the existence of three classes of web servers: very fast, fast and slow. The very fast class contains Nweb, with Null HTTPD and Apache coming close, respectively. The fast class contains the Apache Tomcat web server, which is 30% slower than its non-services-enabled counterpart, and Awhttpd. Finally, the slow class contains the Jetty web server, which is at least 8-10 times slower than all the others. We observe very large service processing times in the case of the Jetty web server, compared to the other five servers. We note that the Jetty web server is the only one using the Java platform, and that the Java Virtual Machine used during our tests was

non-commercial, possibly providing less optimizations. In addition, it is possible that during the testing process of the Jetty web server, the PlanetLab machines used for testing may have been extra loaded.

The web server achieving the smallest average service processing time was Null HTTPD, followed by Nweb, but the web server obtaining the minimum response time among all the requests is Apache. Looking at the variability of the service processing time, the observed standard deviation lies within 10% of the average, for each server. However, the maximum response time outliers range from 10-15 times higher than the average (e.g., NWeb and Apache) to 20-35 times (e.g., Apache Tomcat, Awhttpd). We conclude that, for the selected test scenario, NWeb and Apache are the best performers, followed by Null HTTPD, Apache Tomcat, and Awhttpd (with lower performance or robustness), and then, at some distance, Jetty. Table 2 presents the statistical values for the time to job completion (TTJC) of the six web servers we tested. The average TTJC is higher than the average service processing time due to the workload structure and of the environment performance (notably, due to failures). The results based on TTJC measurement seem to be consistent with our previous conclusions: Apache, Nweb and Null HTTPD achieved the best performance for this test scenario.

Table 3: TTJF for the six web servers (in seconds))

Web Server	Average (Standard Deviation)	Minimum	Maximum	Weighted Average
Apache	No Failures	-	-	-
Null HTTPD	2.7893 (0.000)	0.0000	5.5786	2.7893
Apache Tomcat	No Failures	-	-	-
Nweb	No Failures	-	-	-
Jetty	1.4840 (0.000)	0.000	17.8760	1.4840
Awhttpd	No Failures	-	-	-

Table 3 presents the statistical values for the time to job failure (TTJF) of the six web servers we tested. Analyzing the Failure metric, we notice that in the case of NullHTTPD and Jetty, some failures did occur. We concluded that all of these failures occurred because the requests exceeded the allotted time of 25 seconds. This could have happened for two reasons: either the machine on which the failure occurred was too loaded and the request was delayed, or the machine on which the web server was running became too loaded. Ideally, we would not want the machines on which the testers were running to become too loaded, but we have little control over the load of the machines which are part of PlanetLab.

Our tests show also that ServMark can be used for testing fine-grained services in a wide-scale environment. We have met the main requirements for testing in real grid environments, except for the representative workload generation (see Section 2), which was beyond the scope of this work; however, we have shown in Section 3.1 and in [18] that representative workloads of high complexity can be generated with ServMarks components. The test parameters we chose (20 testers and 100 queries per tester) were large enough to make good use of the resources available at the testing nodes. The testers were fault-tolerant, in the sense that they automatically detected and stopped the blocked testing routines. We have used a single-click test deployment. We have also met the practical requirements (see Section 3.2) by implemented mechanisms.

6 Related Work

A significant number of projects have tried to tackle the Grid performance assessment problem from different angles: modeling workloads and simulating their run under various environment assumptions [3, 5, 15], attempting to produce a representative set of grid applications like the NAS Grid Benchmarks [8], creating synthetic applications that can assess the status of grid services like the GRASP project [4] and the Grid Exerciser [1], and creating tools for launching benchmarks/application-specific functionality tests like the GridBench project [13] and the NMI projects [43]. ServMark is the natural complement to these approaches, by offering a much larger application base, more advanced workload modeling features, and the ability to replay existing workload traces. In addition, ServMark can be used for much more than just Grid performance evaluation.

The modeling/simulation approach is almost exclusively based on traces which are now part of the Parallel Workloads Archive. The major hurdle for this approach is to prove the representativeness of simulation results for real grid environments.

[1] The Grid Exerciser (GEx) is available online at http://www.cs.wisc.edu/condor/tools/exerciser/

Frumkin et al. [8] propose a small set of parallel applications as Grid benchmarks. Simple workloads are defined for the applications, in that the running parameters and the order in which the applications are to be run are fixed. The drawbacks of this approach are that the applications are only representative for a restricted research area (here, computational fluid dynamics), make very little use of Grid components (only Grid-enabled MPI and a scheduler), and cannot adapt to the dynamic behavior of Grids (they require fixed resource sizes, and have no fault-tolerance, migration, or check-pointing features).

Chun et al. [4] use a small set of applications specifically designed to test specific aspects of Grids functionality (probes). The applications assume the existence of common Grid components, like a global information system, or a file-transferring service. No attempt to form workloads with these applications is made. Tsouloupas et al. [13] propose a benchmark launching tool. This tool has the ability to launch benchmarks and display their results, and can be coupled with many of the existing HPC benchmarks. However, it has very limited workload modeling features, and cannot replay real traces. NMI [43] facilitates the definition and run of functionality tests. It currently lacks the ability to define complex workloads, specific for performance and scalability testing.

Many studies have investigated the performance of individual Grid services. As an example, Zhang et al. [26] compare the performance of three resource selection and monitoring services: the Globus Monitoring and Discovery Service (MDS), the European Data Grid Relational Grid Monitoring Architecture (R-GMA), and Hawkeye. Their experiment uses two sets of machines (one running the service itself and one running clients) in a LAN environment. The setup is manual and each client node simulates 10 users accessing the service. This is exactly the scenario where ServMark would have proved its usefulness: it would have freed the authors from deploying clients, coordinating them, and collecting performance results, and allow them to focus on optimally configuring and deploying the services to test, and on interpreting performance results.

The Globus Toolkits job submission service test suite [27] uses multiple threads on a single node to submit an entire workload to the server. However, this approach does not gauge the impact of a wide-area environment, and does not scale well when clients are resource intensive which means that the service will be relatively hard to saturate. The Network Weather Service (NWS) [28, 29] is a distributed monitoring and forecasting system. A distributed set of performance sensors feed forecasting modules. There are important differences to ServMark. First, NWS does not attempt to control the offered load on the target service but merely to monitor it. Second, the performance testing framework deployed by ServMark is built on the fly, and removed as soon as the test ends, while NWS sensors aim to monitor services over long periods of time. Similarly, NETI@home [30], Gloperf [31], and NIMI [32] focus on monitoring service or network level performance. NetLogger [33] targets instrumentation of Grid middleware and applications, and attempts to control and adapt the amount of instrumentation data produced in order not to generate too much monitoring data. NetLogger is focusing on monitoring, and requires code modification in the clients; furthermore, it does not address automated client distribution or automatic data analysis. Similarly, the CoSMoS system [34] is geared toward generic network applications.

GridBench [35] provides benchmarks for characterizing Grid resources and a framework for running these benchmarks and for collecting, archiving, and publishing results. While DiPerF focuses on performance exploration for entire services, GridBench uses synthetic benchmarks and aims to test specific functionalities of a Grid node. However, the results of these benchmarks alone are probably insufficient to infer the performance of a particular service. Finally, Web server performance has been a high-interest topic of recent research [36,37]. The Wide Area Web Measurement (WAWM) Project designs an infrastructure distributed across the Internet allowing simultaneous measurement of web client performance, network performance, and web server performance [36]. Banga et al. [37] measure the capacity of web servers under realistic loads. Both systems could have benefited from a generic framework such as ServMark.

7 Conclusion and Ongoing Work

In this paper we have presented ServMark, a distributed system for testing Grid environments and Grid and web services. We have described its design and have successfully implemented the system. The implementation was tested first on DAS and then, using PlanetLab to deploy the testers, we have evaluated the performance of six Web servers. We have shown how ServMark can fulfill the main requirements for testing in real grid environments: generate realistic workloads, provide accurate testing, be scalable and reliable, and provide hooks for extension (through plug-in mechanisms). We have also shown that in practice ServMark can be easily used used for completely automated testing.

Currently, we are working on improving ServMark in several directions. First, we are trying to improve the

interface between the user and the ServMark controller, for more complex testing scenarios. Second, we are thinking about alternative ways to send the information from the testers to the controller, i.e., through configurable push/pull mechanisms. Third, we are working towards making ServMark a more fault-tolerant grid service.

Availability

The ServMark package is developed jointly by the Delft University of Technology, University of Muenster, University of Chicago, University of British Columbia, and Politehnica Univeristy of Bucharest. ServMark is freely available from its Globus Incubator project homepage: http://dev.globus.org/wiki/Incubator/ServMark

Acknowledgements

This research work is carried out 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). This work was also supported by the EU-NCIT NCIT leading to EU IST Excellency project, EU FP6-2004-ACC-SSA-2.

References

[1] H. E. Bal et al. The distributed ASCI supercomputer project. Operating Systems Review, 34(4):76-96, October 2000.

[2] F. Berman, A. Hey, and G. Fox. Grid Computing: Making The Global Infrastructure a Reality. Wiley Publishing House, 2003.

[3] A. I. D. Bucur and D. H. J. Epema. Trace-based simulations of processor co-allocation policies in multiclusters. In Proc. of the 12th IEEE HPDC, pages 70-79. IEEE Computer Society, 2003.

[4] G. Chun, H. Dail, H. Casanova, and A. Snavely. Benchmark probes for grid assessment. In IPDPS. IEEE Computer Society, 2004.

[5] C. Ernemann, V. Hamscher, U. Schwiegelshohn, R. Yahyapour, and A. Streit. On advantages of grid computing for parallel job scheduling. In CCGRID, pages 39-49. IEEE Computer Society, 2002.

[6] C. Ernemann, B. Song, and R. Yahyapour. Scaling of workload traces. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, JSSPP, volume 2862 of LNCS, pages 166–182. Springer, 2003.

[7] D. G. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. In D. G. Feitelson and L. Rudolph, editors, JSSPP, volume 1459 of LNCS, pages 1-24. Springer, 1998.

[8] M. Frumkin and R. F. V. der Wijngaart. Nas grid benchmarks: A tool for grid space exploration. Cluster Computing, 5(3):247-255, 2002.

[9] H. Li, D. Groep, and L. Wolters. Workload characteristics of a multi-cluster supercomputer. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, JSSPP, LNCS, vol.3277, pages 176-194. Springer, 2004.

[10] H. Mohamed and D. Epema. Experiences with the koala co-allocating scheduler in multiclusters. In Proc. Of the 5th IEEE/ACM Int'l Symp. on Cluster Computing and the GRID (CCGrid2005), May 2005.

[11] W. Smith, I. Foster, and V. Taylor. Predicting application run times with historical information. J. Parallel Distrib. Comput., 64(9):1007-1016, 2004.

[12] A. Snavely, G. Chun, H. Casanova, R. F. V. der Wijngaart, and M. A. Frumkin. Benchmarks for grid computing: a review of ongoing efforts and future directions. ACM SIGMETRICS Perform. Eval. Rev., 30(4):27-32, 2003.

[13] G. Tsouloupas and M. D. Dikaiakos. GridBench: A workbench for grid benchmarking. In P. M. A. Sloot, A. G.Hoekstra, T. Priol, A. Reinefeld, and M. Bubak, editors, EGC, volume 3470 of LNCS, pages 211-225. Springer,

2005.

[14] R. V. van Nieuwpoort, J. Maassen, G. Wrzesinska, R. Hofman, C. Jacobs, T. Kielmann, and H. E. Bal. Ibis: a flexible and efficient java-based grid programming environment. Concurrency & Computation: Practice & Experience., 17(7-8):1079-1107, June-July 2005.

[15] A. M. Weil and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel Distrib. Syst., 12(6):529-543, 2001. [16] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. ACM SIGMOD Rec., 34(3):44-49, 2005.

[16] C. Dumitrescu, I. Raicu, M. Ripeanu, I. Foster, DiPerF: An automated DIstributed PERformance testing Framework, In Proc. of the 5th IEEE GRID Workshop, 2004.

[17] A. Iosup, D.H.J.Epema, GrenchMark: A Framework for Analyzing, Testing, and Comparing Grids, In Proc. of the 6th IEEE/ACM Int'l Symposium on Cluster Computing and the Grid (CCGrid'06).

[18] L. Peterson, T. Anderson, D. Culler, T. Roscoe, A Blueprint for Introducing Disruptive Technology into the Internet, The First ACM Workshop on Hot Topics in Networking (HotNets), October 2002.

[19] A. Bavier et al., Operating System Support for Planetary-Scale Services, Proceedings of the First Symposium on Network Systems Design and Implementation (NSDI), March 2004.

[20] Grid2003 Team, The Grid2003 Production Grid: Principles and Practice, 13th IEEE Intl. Symposium on High Performance Distributed Computing (HPDC-13) 2004.

[21] The Globus Alliance, www.globus.org, Last visited: 10 November 2006.

[22] Foster I., Kesselman C., Tuecke S., The Anatomy of the Grid, International Supercomputing Applications, 2001.

[23] I. Foster, C. Kesselman, J. Nick, S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service

[24] Infrastructure WG, Global Grid Forum, June 22, 2002.

[25] The Globus Alliance, WS GRAM: Developer's Guide, http://www-unix.globus.org/toolkit/docs/3.2/gram/ws.

[26] X.Zhang, J. Freschl, J. M. Schopf, A Performance Study of Monitoring and Information Services for Distributed Systems, Proceedings of HPDC-12, June 2003.

[27] The Globus Alliance, GT3 GRAM Tests Pages, http://www-unix.globus.org/ogsa/tests/gram.

[28] R. Wolski, Dynamically Forecasting Network Performance Using the Network Weather Service, Journal of Cluster Computing, Volume 1, pp. 119-132, Jan. 1998.

[29] R. Wolski, N. Spring, J. Hayes, The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, Future Generation Computing Systems, 1999.

[30] Charles Robert Simpson Jr., George F. Riley. NETI@home: A Distributed Approach to Collecting Endto-End Network Performance Measurements. PAM 2004.

[31] C. Lee, R. Wolski, I. Foster, C. Kesselman, J. Stepanek. A Network Performance Tool for Grid Environments, Supercomputing '99, 1999.

[32] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis. An architecture for large-scale internet measurement. IEEE Communications, 36(8):4854, August 1998.

[33] D. Gunter, B. Tierney, C. E. Tull, V. Virmani, On-Demand Grid Application Tuning and Debugging with the NetLogger Activation Service, 4th International Workshop on Grid Computing, Grid2003, November 2003.

[34] Ch. Steigner and J. Wilke, Isolating Performance Bottlenecks in Network Applications, in Proceedings of the International IPSI-2003 Conference, Sveti Stefan, Montenegro, October 4-11, 2003.

[35] G. Tsouloupas, M. Dikaiakos. GridBench: A Tool for Benchmarking Grids, 4th International Workshop on Grid Computing, Grid2003, Phoenix, Arizona, November 2003.

[36] P. Barford ME Crovella. Measuring Web performance in the wide area. Performance Evaluation Review, Special Issue on Network Traffic Measurement and Workload Characterization, August 1999.

[37] G. Banga and P. Druschel. Measuring the capacity of a Web server under realistic loads. World Wide Web Journal

(Special Issue on World Wide Web Characterization and Performance Evaluation), 1999.

[38] N. Minar, A Survey of the NTP protocol, MIT Media Lab, December 1999, [Online] Available: http://xenia.media.mit.edu/ñelson/research/ntp-survey99, November 2006.

[39] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, A Resource Management Architecture for Metacomputing Systems, IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pg. 62-82, 1998.

[40] Feitelson, D.G., Rudolph, L., Schwiegelshohn, U., Sevcik, K.C., Wong, P.: Theory and Practice in Parallel Job Scheduling. In Feitelson, D.G., Rudolph, L., eds.: Proc. of the 3rd Intl. Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP). Volume 1291 of Lecture Notes in Computer Science., Geneva, Springer-Verlag (1997) 134.

[41] A. Iosup, D.H.J. Epema, C. Franke, A. Papaspyrou, L. Schley, B. Song, R. Yahyapour, On Grid Performance Evaluation using Synthetic Workloads, In The 12th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), held in conjunction with SIGMETRICS'06, Jun 26, 2006, Saint Malo, FR.

[42] A. Iosup, C. Dumitrescu, D.H.J. Epema, H. Li, L. Wolters, How are Real Grids Used? The Analysis of Four Grid Traces and Its Implications, The 7th IEEE/ACM International Conference on Grid Computing (Grid), Barcelona, September 28-29, 2006.

[43] Andrew Pavlo, Peter Couvares, Rebekah Gietzel, Anatoly Karp, Ian D. Alderman, and Miron Livny, The NMI Build & Test Laboratory: Continuous Integration Framework for Distributed Computing Software, The 20th USENIX Large Installation System Administration Conference (LISA), Washington, D.C., December 38, 2006 (accepted)

[44] H.H. Mohamed and D.H.J. Epema, An Evaluation of the Close-to-Files Processor and Data Co-Allocation Policy in Multiclusters, CLUSTER 2004, IEEE Int'l Conference Cluster Computing 2004, September 2004.

[45] H.H. Mohamed and D.H.J. Epema, Experiences with the KOALA Co-Allocating Scheduler in Multiclusters, Proc. of the 5th IEEE/ACM Int'l Symp. on Cluster Computing and the GRID (CCGrid2005), Cardiff, pp. 784-791, May 2005.

[46] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet", Proceedings of the First ACM Workshop on Hot Topics in Networking (HotNets), October 2002.

[47] I. Foster, et al., "The Grid2003 Production Grid: Principles and Practice", 13th IEEE Intl. Symposium on High Performance Distributed Computing, 2004.