A Performance Oriented Grid Monitoring Architecture

S. De Smet, P. Thysebaert*, B. Volckaert, M. De Leenheer, D. De Winter, F. De Turck†, B. Dhoedt, P. Demeester
Department of Information Technology, Ghent University - IMEC
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium
Email: stijn.desmet@intec.ugent.be
*Research Assistant of the Fund for Scientific Research - Flanders (F.W.O.-V., Belgium)
†Postdoctoral Fellow of the Fund for Scientific Research - Flanders (F.W.O.-V., Belgium)

***Abstract*— Resource state monitoring is a critical component of any Grid Management Architecture, providing Grid scheduler, job/execution manager and state estimation components with accurate information about network, computational and storage resource status. Without up-to-date monitoring information, intelligent scheduling decision making would be a near-impossible task. In this paper we describe a scalable, portable and non-intrusive Grid Monitoring Architecture whose implementation decisions were made with performance in mind. We compare it to well-known Grid monitoring systems, and measure our platform's performance to the Globus MDS2.2 and the Globus 3.2 web services Information Service (WS-IS) performance.**

Keywords: Grid Monitoring, GMA, Globus MDS

I. Introduction

A Grid provides a uniform interface to a collection of heterogeneous, geographically distributed resources. These resources are dynamic in nature (i.e. resources can join/part from the Grid, hardware failures can occur, etc.) and every resource has its own specific properties and status information. This information can in turn be used by the Grid scheduling entity; in a distributed computing environment, one of the key components necessary to be able to perform effective job scheduling (and thereby improve resource utilization), is a resource monitoring architecture. The requirements for such a monitoring system are in no specific order: efficiency, scalability, portability and extensibility.

These requirements have been recognized by the Global Grid Forum (which acts as the governing body for Grid standardization), and this resulted in the conception of a reference architecture for feasible Grid Monitoring systems, dubbed 'Grid Monitoring Architecture' (GMA [1]).

In this paper, we present a feature-rich, highly extensible yet performance oriented Grid monitoring platform, developed according to the GMA specifications. Important features include configurable caching mechanisms, non-intrusiveness, support for third party sensor plugins and an intuitive GUI offering one-click visualization. We discuss the technology decisions that were made when developing this platform, and compare its performance to the Globus Monitoring and Discovery Service [2].

This paper continues as follows: Section II gives an overview of important related work and highlights the differences with our monitoring system. A high-level description of the constituent components is given in section III, while technical decisions made during implementation are discussed in section IV. Our test results are presented in section V, followed by a brief look at future work in section VI and conclusions in section VII.

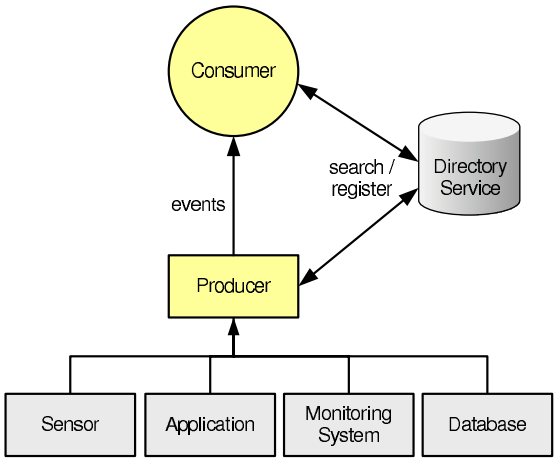

Fig. 1. Grid Monitoring Architecture overview

II. Related Work

The Grid Monitoring Architecture, as defined by the GGF, consists of three important components: producers, consumers and a directory service (see figure 1). The directory service stores the location and type of information provided by the different producers, while consumers typically query the directory to find out which producers can provide their needed event data (after which they contact the producers directly). Producers in turn can receive their event data from a variety of providers (software/hardware sensors, applications, whole monitoring systems, databases, etc.). The GMA does not specify the underlying data models or protocols that have to be used.

Multiple monitoring architectures for distributed computing systems have already been successfully deployed. Not all of them follow the guidelines set by the GMA (e.g. Condor's

Hawkeye [3] which does not support a decentralized architecture), and some are geared towards monitoring one single resource type (e.g. Remos [4], focusing on network parameters). Below we present some notable Grid monitoring platforms with an architecture similar to our framework, and point out the architectural or implementation-specific differences with our GMA implementation. For a complete overview of Grid monitoring tools we refer to [5].

GMA-compliant Grid monitoring systems include the European DataGrid's Relational Grid Monitoring Architecture R-GMA [6] and GridRM [7]. R-GMA offers a combined monitoring and information system using a Relational Database Management System as directory service and monitoring data repository (this approach offers the possibility to formulate complex queries on the monitored data i.e. it allows to locate monitoring components and retrieve the data they offer using standard SQL statements). The implementation is based on Java servlet technology (using the Tomcat servlet container), trading performance for portability and limited software dependencies.

GridRM is an open source two-layer Grid monitoring framework, the upper layer being structured according to the GMA. This upper layer connects the per-site monitoring systems (the lower layer) in a scalable way. Like R-GMA, GridRM makes use of Java and SQL to query data producers. Currently, GridRM's directory service (containing info on the location of the different resource status providers) can be a bottleneck and/or single point of failure, but work is underway to remedy this problem.

MDS2 is the Globus Toolkit (version 2) Monitoring and Discovery Service, and although MDS development was started before the GMA architectural reference appeared, it can still be seen as a GMA implementation. MDS2 only supports latest-state queries, making it mandatory for the consumers to actively retrieve status information from the GRIS (MDS2 component offering producer-like functionality). In addition, MDS2 does not offer visualization features. In section V, we have compared some performance characteristics of MDS2 and our Grid Monitoring Architecture. An extensive comparison of MDS2 against other monitoring frameworks has already been carried out and presented in [8]. It was shown that MDS2 outperforms (i.e. exhibits lower response times and better scalability) the other frameworks mentioned in most use cases. Therefore, we have only compared our platform's performance to that of MDS2 and its successor, the web services based Information Service [9] from the Globus 3.2 Toolkit. For ease of comparison, we have evaluated this performance using similar tests as described in [8].

III. Grid Monitoring Framework Components

In figure 2 a sample setup of our framework featuring its constituent components is shown. Each component's function is detailed below.

A. Sensor

Every resource to be monitored has at least one sensor attached to it. Each sensor can monitor different load properties of a single resource. For instance, we have implemented a CPU sensor capable of monitoring CPU load, idle time, frequency and time spent executing user processes. The actual values are then communicated to one or more producers. This list of producers (and conversely, the list of sensors that is allowed to communicate with each producer) and the frequency at which each producer is contacted are readily found in the directory service. The only configurable parameters of the sensor are the location and authentication parameters to query the directory service. When a sensor registers itself with a producer, a permanent data connection (over which load values are pushed) is established, to avoid connection setup overhead on every update. The currently implemented sensors can provide detailed status information on CPU, memory, swap, disk and network usage.

B. Producer

Producers register themselves with the directory service and publish the type of information (aggregated from the sensors that report to the producer) they provide. This data can be queried by authorized consumers (using a request/reply model) or can be pushed to authorized consumers using a subscription/notification event-based model. In this way, producers correspond to the producer components defined in the GMA. Each producer has its own cache, storing a configurable number of status updates for each registered sensor. Consumers can retrieve either the last known status update from a specific sensor, or historic data from the producer's cache. Furthermore, producers provide a limited number of statistical operations (average, minimum, maximum, standard deviation, etc.) on cached data. Sensor failures (e.g. resource went offline) can be detected and a failure notification will be sent to consumers who were interested in this sensor's status.

C. Directory Service

The directory service contains information on the registered producers (and their respective offered status information), the producer-sensor mappings and producer access control lists. It is queried by producers and sensors to retrieve these mappings, and by consumers to find a set of providers matching some criterium. A web services enabled directory service management component has been implemented.

D. Consumer

Following the best practices defined by the GMA, consumers query the directory service to find out about producers capable of delivering the desired monitoring data. Consumers proceed by directly contacting these producers, either to retrieve data using a request/reply pattern or to register themselves in order to retrieve future data using an event-based subscription model. Several useful consumers have been implemented, the most important being an archiving consumer (storing data in a relational database and offering a GUI view of historical

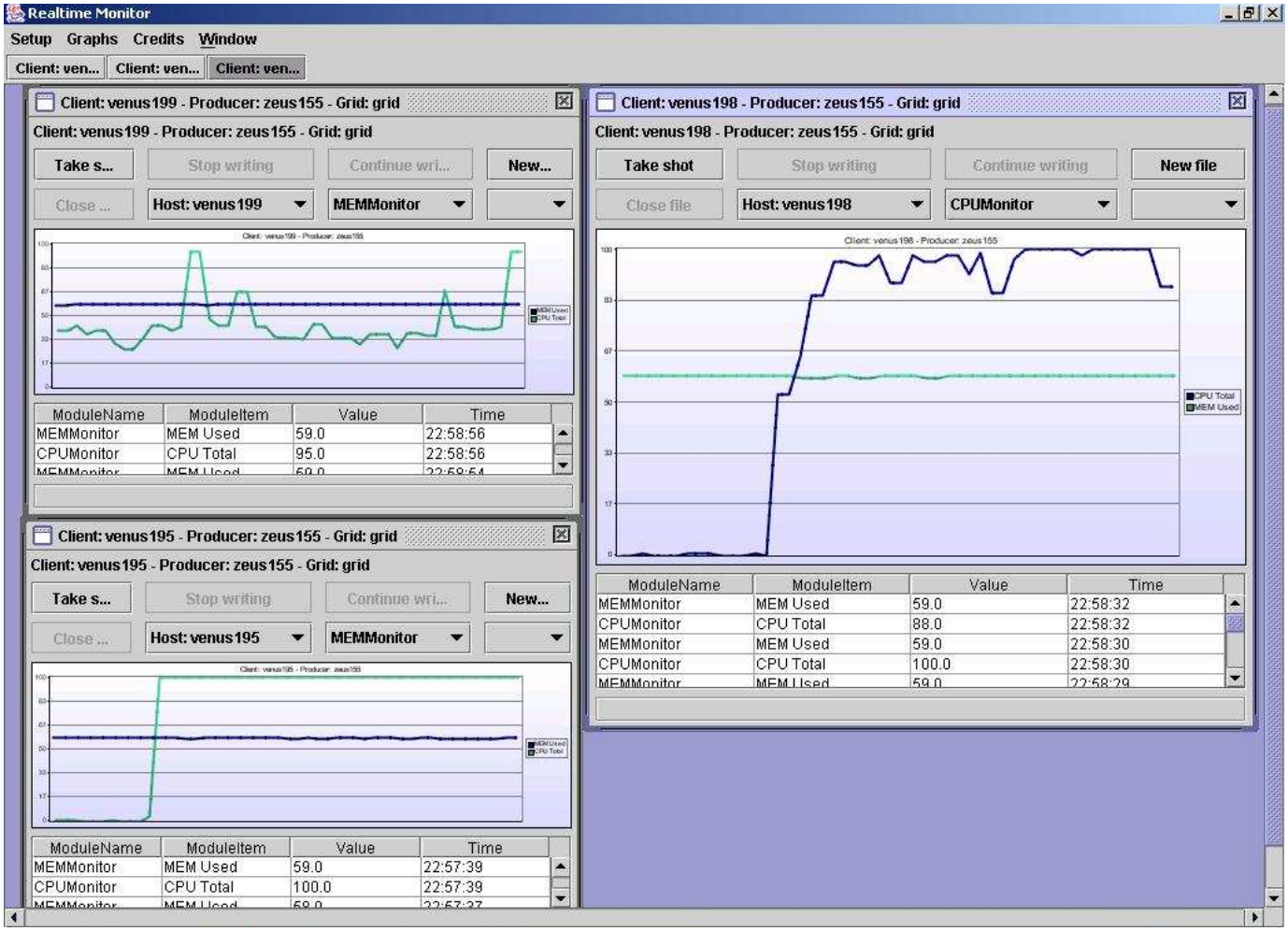

Fig. 3. Real-time Java visualization agent

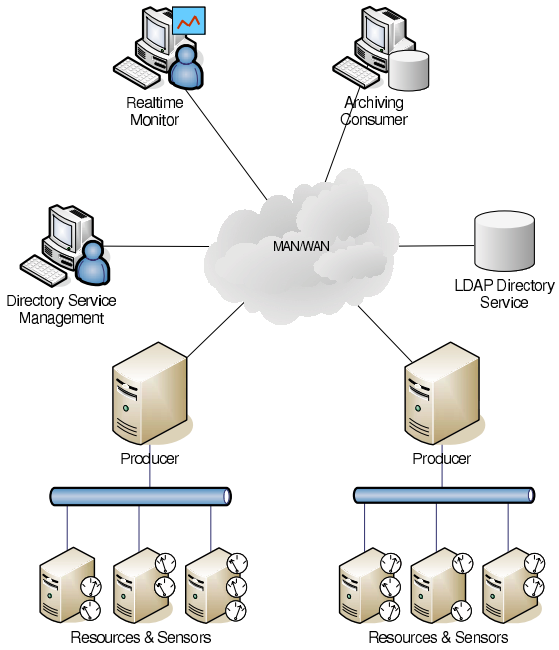

Fig. 2. Grid Monitoring Framework

data), a real-time Java-based visualization agent (see figure 3) and a Monitoring Server which interfaces directly with Grid Schedulers or Information Services to aid in schedule calculations.

IV. Technology Analysis

Our GMA-compatible Grid Monitoring Framework was designed to achieve good performance while maintaining a high level of portability.

Our main requirement has driven us to the use of the C++ language in the implementation of the different components. While natively compiled components are known to be more performant than for instance Java bytecode running in a virtual machine, the C++ standard library does not offer cross-platform solutions for various vital application patterns such as networked communication and multiple threads of execution. Therefore, the need arises to use a portable and performant C++ middleware platform offering elegant implementations of these high-level features. In our monitoring architecture, we have decided to use the Adaptive Communication Environment (ACE [10]). It offers cross-platform multithreading, and its

'reactor' concept allows for the easy implementation of event-based (including network events) applications. Furthermore, we make use of the 'Acceptor/Connector' pattern offered by ACE to open networked communication channels between producers and sensors.
Whenever a sensor needs to send statistical data to the producers they are registered with, the data is sent in CORBA Common Data Represenation (CDR) format, offering a portable, network optimised way of communicating.
The resource monitoring data gathered by the sensors is obtained through the GTop library [11], a portable C/C++ library offering access to performance values related to system resources. Each resource is monitored by a sensor plug-in, which is essentially a shared library. The plug-in approach is enabled by the fact that ACE features cross-platform dynamic loading of shared libraries.
Our directory service is essentially a decentralized LDAP [12] directory server. While the expressive power of LDAP queries does not match that of e.g. an SQL query over a relational database, LDAP allows for performant look-ups in a write-once, read-many context.
Producers provide a WSDL [13] interface through which they are contacted by consumers using SOAP. SOAP is an XML-based RPC protocol which can be transferred over HTTP; as such, the use of SOAP allows for the easy integration of our monitoring architecture in a web services-based Grid environment. In our implementation we used gSOAP as reported on in [14].
An overview of the communication methods used between the various components of our Grid Monitoring Architecture is given in table I. It should be noted that SSL encryption is possible for both SOAP-over-HTTP and LDAP communication. This allows access control through the use of user and server certificates. The data updates between sensors and producers can be secured by enabling an SSL socket adapter in these components.

V. Results

A. Testbed Setup

Six machines (AMD Duron 750Mhz, 64MB RAM) have a sensor deployed on them, and four other machines (Intel P4 3GHz, 1GB RAM) carry one producer each; two producers have two sensors registered with them, and the other two have one sensor registered. An OpenLDAP directory server was deployed on a separate machine featuring dual Xeon processors (2.8Ghz, 1GB RAM). Lastly, the consumers (implemented as concurrent threads) used in the tests are located on a second dual Xeon machine. All machines are interconnected through a 100Mbps switched Ethernet LAN; this setup allows us to evaluate intrusiveness and scalability of the different components without suffering significant network bottlenecks.
MDS2 was deployed as follows: a GRIS/GIIS pair ran on the Intel P4 machines (instead of the producers), sensors were replaced with GRIS components whose monitoring data was cached by the Intel P4 GIIS. Our LDAP directory service was replaced by a GIIS (on the dual Xeon machines) connected to the lower level GIISs. Consumers in our MDS tests were spawned from the same machine as our first tests.
The GT 3.2 WS-IS (web services based Information Services) was deployed on the AMD Duron and Intel P4; on the AMD Duron machines, the WS-IS was configured to submit its data to a Pentium 4 machine (which used to run a producer). Again, consumers were spawned from a dual Xeon machine.

B. Metrics

Two metrics were used to evaluate component performance: *throughput* and *response time*. During a 10 minute period, "users" submitted blocking queries to the component under investigation, while waiting for 1 second between successive queries. The throughput was then taken to be the number of queries handled by the component per time unit; the response time is the average amount of time taken to process 1 user query.

C. Intrusiveness

The intrusiveness of our producer components is shown in figure 4, and compared to the load generated by the MDS GRIS. The network traffic generated was monitored using the SCAMPI [15] multi-gigabit monitoring framework (partly developed at our research institution). The CPU load is the average (over the 600 second interval) one minute CPU load average, as measured by *uptime* and related tools.
The higher network load generated between our producer and the consumers stems from the use of the SOAP-over-HTTP XML-based communication mechanism (note that we did not enable *zlib* compression). Note that the web services approach used by GT 3.2 imposes a network load comparable to that of our architecture. However, beyond 100 concurrent users, the GT3.2 Information Services do not scale well, which explains their apparent low network intrusiveness. In analogy, the CPU load generated by the WS-IS seems to degrade with increasing number of concurrent users, but this is slightly deceptive: as the WS-IS does not scale beyond 100 users, it is no longer able to keep up with an appropriate pace of query response generation from this point on. It should be noted that the WS-IS framework is the only monitoring and information framework in these tests which is completely web services based, trading performance for a standards based interface.

D. Directory Service Scalability

The throughput and response times for directory service queries were compared (figure 5) to those obtained for queries against the MDS GIIS (operating on cached data). Average response times are lower for our directory service; however, both our directory service and the MDS GIIS don't scale well beyond 450 concurrent users in this scenario on the given hardware. It should be noted, however, that a GIIS typically contains more data (including cached monitoring data) than our directory service, which only stores configuration data, never monitoring data (this should be requested straight from a producer or from an archiving consumer). This led to a bigger result set when the GIIS was queried. In addition, our

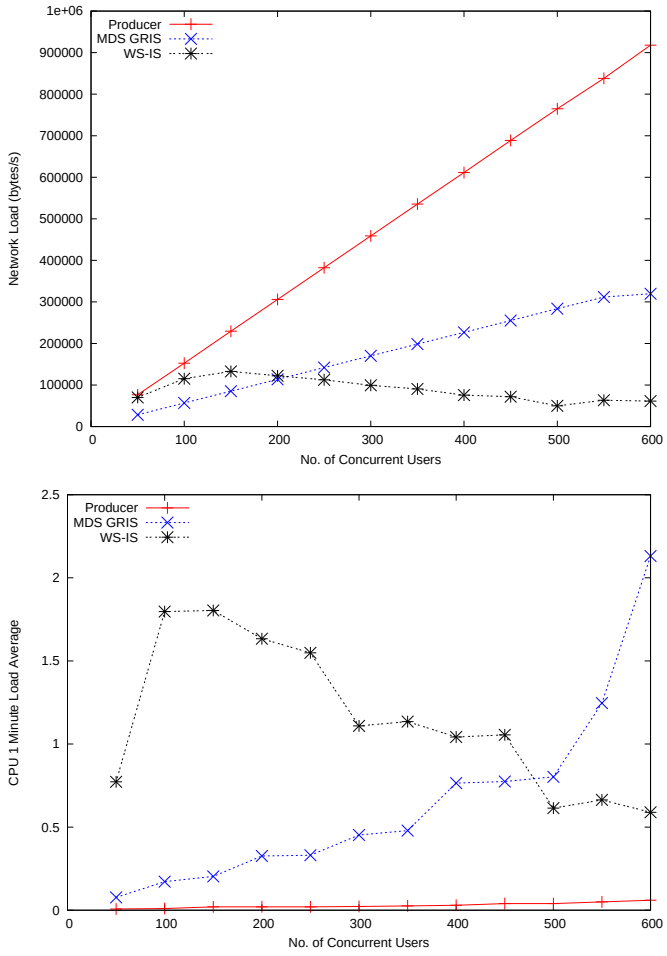

Fig. 4. Producer vs. MDS GRIS vs. WS-IS Network/CPU Intrusiveness

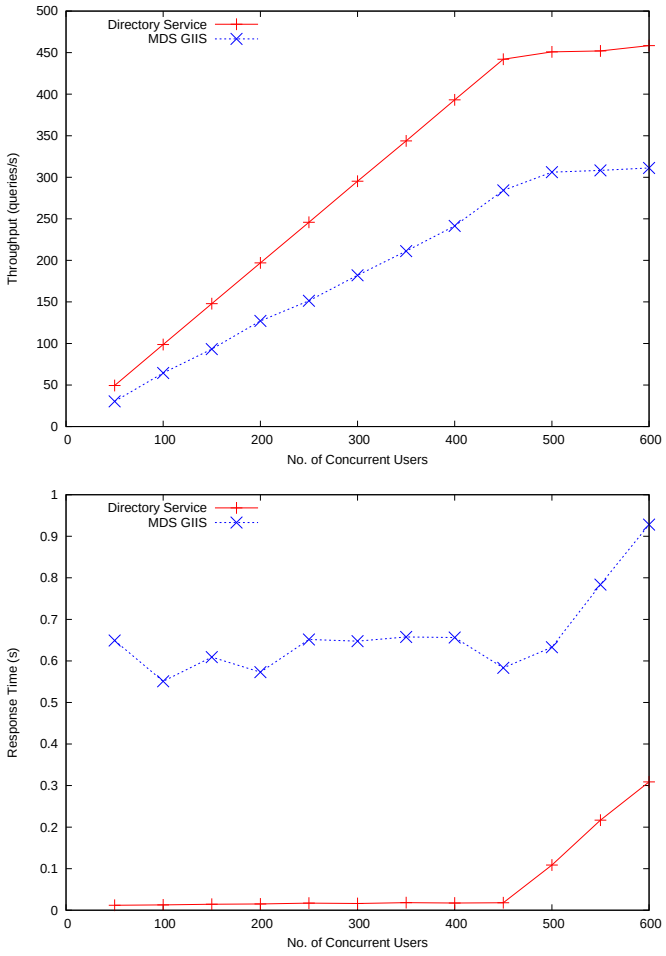

Fig. 5. Directory Service vs. MDS GIIS Scalability

directory service is a plain OpenLDAP server, without module extensions. We chose not not to show results for the GT 3.2 because of the absence of a dedicated component offering functionality which corresponds to our directory service or the MDS2.2 GIIS.

E. Producer Scalability

In figure 6, we have compared producer scalability with increasing number of concurrent users for both our framework's producers and MDS GRIS components. Again, only cached data was requested from the MDS GRIS; due to the use of a push-model our framework's producers always contain up-to-date information, while the GRIS would have to invoke information providers to refresh its data. We measured only small differences between MDS GRIS and our producer (best visible on the response time graph). Beyond 550 concurrent users and using the given machines, MDS GIIS performance started to degrade. We also compared our producer scalability to the scalability of the GT 3.2 Information Service. Again, it is clear that our installation of GT3.2 does not scale well beyond 100 concurrent users (the GT 3.2 results even forced us to use a logarithmic scale in the right part of figure 5, which shows that response times differ by as much as a factor of 100).

VI. Future Work

Because manually deploying sensor plug-ins on multiple resources is tedious, our main focus in the near future will be on platform bootstrapping components. The main idea here is that these bootstrapping components are loaded on a resource as soon as this resource is brought on-line ("installed" when the resource is a computer). Their main objective is to record this resource's presence in the directory service, auto-install suitable (based on the resource's class) sensor plug-ins and register with a set of producers (based on the type of monitoring data offered and the resource's location).

A second topic of interest is the development of proxy producers (remember that sensors and producers communicate using ACE socket streams) which allows to collect sensor data from other monitoring platforms' "information providers" and use these data in our framework.

Lastly, an additional effort is required to adapt our implementation (in particular the interfaces implemented by the various components) in order to comply with the Web Services

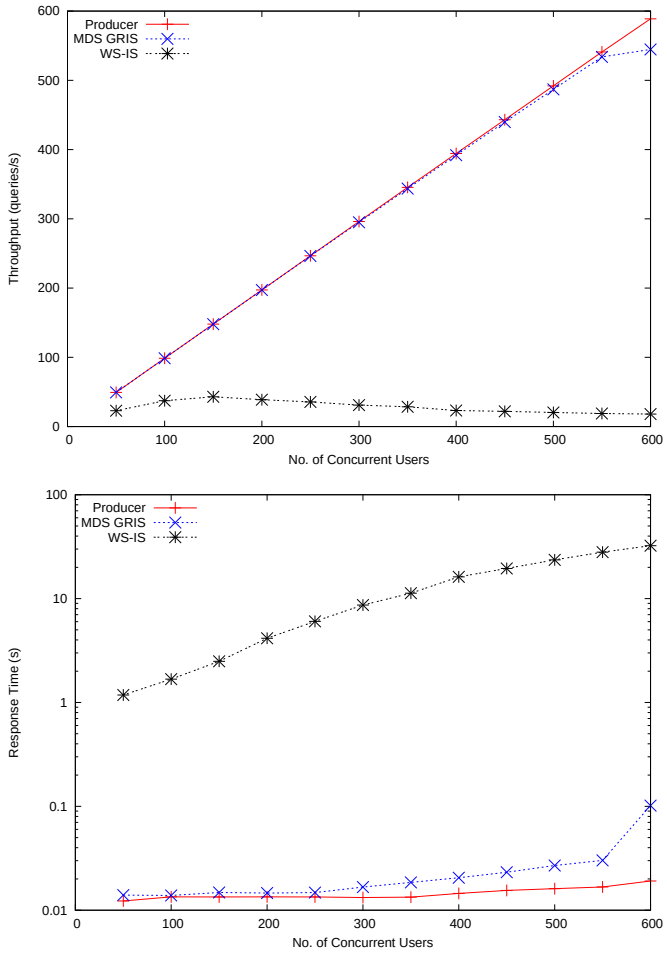

Fig. 6. Producer vs. MDS GRIS vs. WS-IS Scalability

Resource Framework [16] (unifying Grid and Web Services communities) specifications.

VII. Conclusions

We presented a performant and portable implementation of the GGF Grid Monitoring Architecture. Performance was obtained through the use of C++ as base implementation language; portability then dictated the use of appropriate middleware for which we chose the Adaptive Communication Environment. We compared the performance of our implementation to that of the Globus MDS2 system and its successor (the GT 3.2 web services based Information Service using the default supplied OGSI-compliant container), with good results in terms of throughput and response time, both for producers and the directory service. Multiple ready-to-use consumers (including real-time visualization) have been implemented. Within our implementation, heavy use is made of SOAP for consumer-producer communication. With the apparent convergence of the web services and grid communities in mind, we expect this to ease the deployment of our implementation in a services enabled grid environment.

Acknowledgment

B. Volckaert would like to thank the Institute for the Promotion of Innovation through Science and Technology in Flanders.

References

[1] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski, "A grid monitoring architecture," `http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-3.pdf`, 2002.

[2] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in *Proc. of the 10th IEEE International Symposium on High-Performance Distributed Computing*, 2001.

[3] "Hawkeye website," `http://www.cs.wisc.edu/condor/hawkeye/`.

[4] B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok, "A resource query interface for network-aware applications," in *Proc. of the 7th IEEE Symposium on High-Performance Distributed Computing*, 1998.

[5] M. Gerndt, R. Wismuller, Z. Balaton, G. Gombas, P. Kacsuk, Zs. Nemeth, N. Podhorszki, H.-L.Truong, T. Fahringer, M. Bubak, E. Laure, and T. Margalef, "Performance tools for the grid: State of the art and future," *Research Report Series*, vol. 30, 2004.

[6] A. Cooke, A.Gray, L. Ma, W. Nutt, J. Magowan, P. Taylor, R. Byrom, L. Field, S. Hicks, and J. Leake et al, "R-gma: An information integration system for grid monitoring," in *Proc. of the 11th International Conference on Cooperative Information Systems*, 2003.

[7] M.A. Baker and G. Smith, "Gridrm: A resource monitoring architecture for the grid," in *Proc. of the 3rd International Workshop on Grid Computing*, Springer-Verlag, Ed., 2002.

[8] X. Zhang, J.L. Freschl, and J. Schopf, "A performance study of monitoring and information services for distributed systems," in *Proc. of the 12th IEEE International Symposium on High-Performance Distributed Computing*, 2003.

[9] "WS-IS website," `http://www-unix.globus.org/toolkit/docs/3.2/infosvcs/ws/key/index.html`.

[10] D.C. Schmidt and S.D. Huston, *C++ Network Programming: Mastering Complexity Using ACE and Patterns*, Addison-Wesley Longman, 2002.

[11] "LibGTop website," `http://www.gnu.org/directory/libs/LibGTop.html`.

[12] "OpenLDAP website," `http://www.openldap.org`.

[13] "Web service definition language (WSDL)," `http://www.w3.org/TR/wsdl`.

[14] R.A. van Engelen and K.A. Gallivan, "The gSOAP toolkit for Web Services and Peer-To-Peer Computing Networks," in *Proc. of the 2nd IEEE International Symposium on Cluster Computing and the Grid*, 2002.

[15] J. Coppens, S. Van den Berghe, H. Bos, E.P. Markatos, F. De Turck, A. Oslebo, and S. Ubik, "SCAMPI: A scalable and programmable architecture for monitoring gigabit networks," in *Proc. of the 1st Workshop on End-to-End Monitoring Techniques and Services*, 2003.

[16] "The WS-Resource Framework," `http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf`.

	Sensor	**Producer**	**Consumer**	**Dir. Service**
Sensor	/	ACE	/	LDAP
Producer	ACE	/	SOAP	LDAP
Consumer	/	SOAP	/	LDAP
Dir. Service	LDAP	LDAP	LDAP	/

TABLE I

COMMUNICATION TECHNOLOGIES