SCALEA-G: a Unified Monitoring and Performance Analysis System for the Grid *

Hong-Linh Truong
Institute for Software Science
University of Vienna
Liechtensteinstr. 22, A-1090 Vienna, Austria
truong@par.univie.ac.at

Thomas Fahringer
Institute for Computer Science
University of Innsbruck
Technikerstr. 25/7, A-6020 Innsbruck, Austria
Thomas.Fahringer@uibk.ac.at

Abstract

In this paper we describe SCALEA-G which is a unified monitoring and performance analysis system for the Grid. SCALEA-G is being implemented as a set of grid services based on the Open Grid Service Architecture (OGSA). SCALEA-G provides an infrastructure for conducting online monitoring and performance analysis of a variety of Grid services including computational and network resources, and Grid applications. Both push and pull model are supported, providing flexible and scalable monitoring and performance analysis for the Grid. Source code and dynamic instrumentation are exploited to perform profiling and tracing of Grid applications. A novel instrumentation request language has been developed to facilitate the interaction between client and instrumentation services. SCALEA-G offers a hierarchical view of performance characteristics which enables both local and global performance analysis. System and application specific metrics are related to each other where possible in order to simplify the detection of Grid performance problems. Moreover, SCALEA-G provides interfaces for external tools and services not only to access performance data and results but also to control the monitoring.

1 Introduction

Grid Monitoring is an important task that provides useful information for several purposes such as performance analysis and tuning, performance prediction, fault detection, and scheduling. Most existing Grid monitoring tools are separated into two distinct domains based on what they are monitoring: *Grid infrastructure monitoring* and *Grid application monitoring*. The lack of combination of two domains in a single system has hindered the user from relating measurement metrics of various sources at different levels when conducting the monitoring and performance analysis. Grid monitoring tools that are able to combine application and system monitoring and performance analysis are crucial as these tools will provide the user a unified view and support the correlation between performance metrics from various sources. In addition, most existing Grid monitoring tools focus on the monitoring and analysis for Grid infrastructure; yet little effort has been done on performance analysis for Grid applications. To date, application performance analysis tools are mostly targeted to conventional parallel and distributed systems (e.g. clusters, SMP machines). As a result, these performance analysis tools do not well address challenges in Grid environment such as scalability, diversity, dynamics and security.

To attack on above-mentioned issues, we are developing a new system namely SCALEA-G (SCALEA Grid-enabled) which is based on our experience of building performance instrumentation, measurement, visualization and analysis for conventional parallel systems [26]. In this paper we describe SCALEA-G, a unified system for monitoring and performance analysis in the Grid. SCALEA-G is based on the concept of Grid Monitoring Architecture (GMA) [1] and being implemented as a set of OGSA-based grid services [13]. SCALEA-G consists of distributed monitoring and performance sensors and sensor manager services, directory service for supporting information publication and discovery, archival services for storing collected monitoring and performance data, instrumentation services and various performance analyzers. It is built up on a set of open protocols, partially reuses existing state-of-the art tools and framework for Grid, and the SCALEA system [26]. SCALEA-G provides an infrastructure of OGSA-based grid services for online monitoring and performance analysis of a variety of Grid services including computational resources, networks, and applications. SCALEA-G's services can easily

*This research is supported by the Austrian Science Fund as part of the Aurora Project under contract SFBF1104.

be deployed and used by multiple users and other services. Both push and pull model proposed in GMA are supported, providing a flexible and scalable way when performing the monitoring and analysis. In SCALEA-G, each kind of monitored data is represented in XML in accompany with an XML schema [30], allowing any client to easily access the data via XPath/XQuery [29, 31].

SCALEA-G supports both source code and dynamic instrumentation for profiling and tracing of Grid applications. A novel instrumentation request language has been devised to facilitate the interaction between client and instrumentation services. The performance behavior of monitored resources, services can be examined and viewed in hierarchical manner, allowing to exploit performance characteristics from local (e.g. within a computational node) to global (e.g among grid sites). System and application specific metrics are related as close as possible, thus increasing the chance to uncover Grid performance problems. In addition, SCALEA-G provides interfaces for external tools to exploit SCALEA-G services as well as to access information provided by SCALEA-G.

The rest of this report is organized as follows: Section 2 presents the architecture of SCALEA-G. In Section 3, we describe SCALEA-G sensors and Sensor Manager Service in details. Section 4 overviews SCALEA-G Client Service. Section 5 describes instrumentation service and instrumentation request language. We then discuss the data delivery, caching and filtering mechanism in Section 6. Security issues in SCALEA-G are presented in Section 7 followed by an overview of the implementation of SCALEA-G in Section 8. Section 9 illustrates experiments and examples in which SCALEA-G is used to perform the monitoring and analysis. We present some related work in Section 10 before going to the Conclusion and Future work in Section 11.

2 SCALEA-G Architecture

SCALEA-G is an open architecture based on OGSA [13] combined with GMA [1]. Figure 1 depicts the architecture of SCALEA-G which includes main services: SCALEA-G's Directory Service, Archival Service , Sensor Manager Service , Instrumentation Service and Client Service and an User Portal. All of them are OGSA-based services/clients and can be deployed on different hosting environments.

SCALEA-G Directory Service is used for publishing and searching information. It stores information about producers and consumers that produce and consume performance data and information about types, characteristics of that data.

SCALEA-G Archival Service is a data repository which is used to store detailed monitoring information as well as performance results collected and analyzed by other components.

SCALEA-G Sensor Manager Service is used to manage sensors that gather and/or measure a variety of kinds of data for monitoring and performance analysis. One Sensor Manager Service can interact with several sensor instances executed on the same physical machine or different machines. Sensor instances will send their collected data to Sensor Manager Service. At the Sensor Manager Service side, the data will be buffered and can optionally be stored in the SCALEA-G Archival Service. Sensor Manager Service is also responsible for processing subscription and query requests from consumers. SCALEA-G Sensor Manager Service is similar to a join Producer and Consumer in GMA model. It implements filtering, forwarding and caching data to/from various destinations/sources.

SCALEA-G Instrumentation Service is used to instrument applications. Instrumentation can be done at source code level or dynamically at runtime of the application.

SCALEA-G Client Service provides interfaces for administrating other SCALEA-G services and accessing data in these services. In addition, it provides facilities for analyzing performance data. Any external tools interesting with data provided by SCALEA-G can access SCALEA-G by using SCALEA-G Client Service.

SCALEA-G User Portal supports the user to conduct online monitoring and performance analysis. It is based on facilities provided by SCALEA Client Service.

SCALEA-G services register information about their service instances with a *Registry Service*. They can be deployed in two cases: 1) as shared services used by multiple users, and 2) as user's own services. In the first case, the administrator of these services will control the user who is allowed to use the services; the user can find these services by looking for their service data in Registry Service. In the second case, only the user who invokes these services can use and administrate the services.

Interactions among SCALEA-G services and clients are divided into *Grid service operation invocations* and *stream data delivery*. Grid service operation invocations are used to perform tasks including controlling activities of services and sensors, subscribing and querying requests for performance data, registering, querying and receiving information of Directory Service. In stream data delivery, a stream channel is used to transfer data (monitored data, performance data and results) between producers (e.g. sensors, Sensor Manager Service) and consumers (e.g. Sensor Manager Service, clients). Data channel is persistent stream connection. Each Sensor Manager Service has only one connection to a consumer for delivering all kinds of data. All the connections are secure sockets based on Grid Security Infrastructure (GSI) and Public Key Infrastructure (PKI). In the following sections, we will describe various services of SCALEA-G in details.

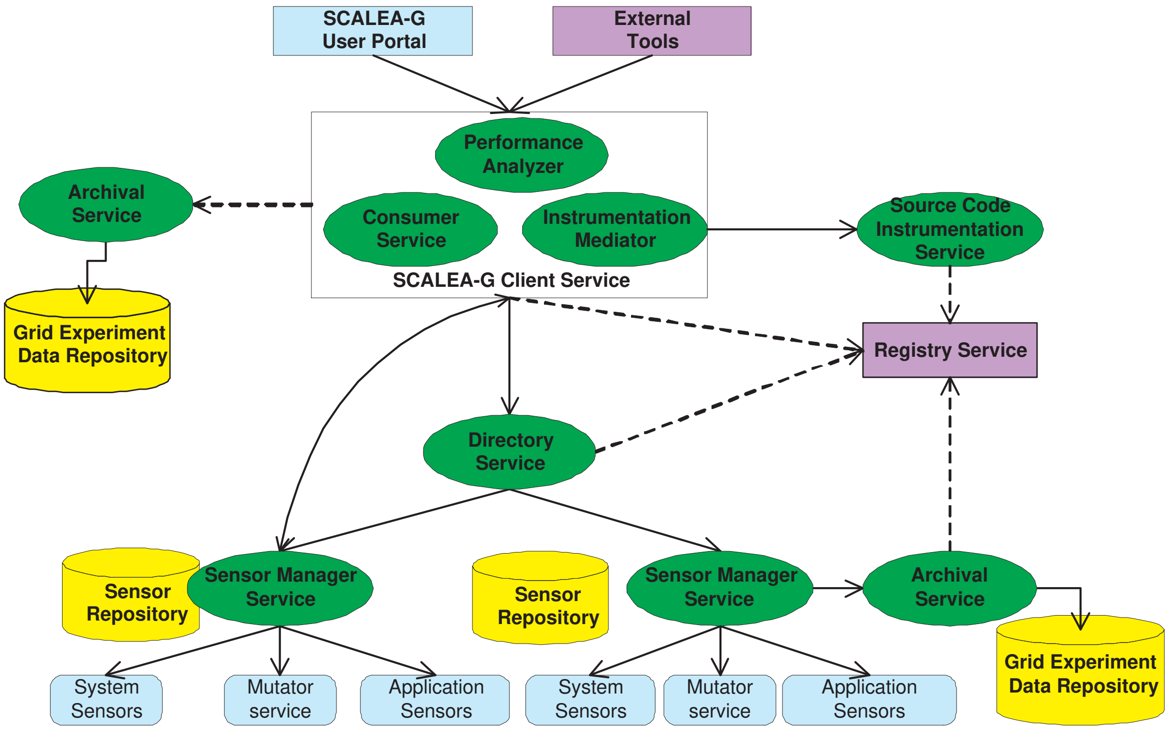

Figure 1. High-level view of SCALEA-G Architecture

3 Sensors and Sensor Manager Service

Sensors are used to capture performance data and to monitor status of Grid services including computational and network resources, applications. SCALEA-G distinguishes two kinds of sensors: *system sensors* and *application sensors*. System sensors are used to monitor and measure the performance of Grid *computational services* (e.g. computational hosts) and *network services* (e.g. network connections). Application sensors embedded in *software services* are used to measure execution behavior of code regions and to monitor events in these services[1].

A sensor can be invoked at different times; each invocation of a sensor is called *sensor instance* referring to a particular instantiation of a sensor at run-time. Sensor instances deliver data they collect, described in XML format, to Sensor Manager Services. Each sensor instance is associated with a lifetime in which the sensor instance is active. Different instances of a sensor can be used to monitor different resources, services and they might require different parameters. For example, a sensor which is used to measure the available bandwidth between two compute nodes in the Grid can have two instances: one used to monitor the bandwidth between node A and B and the other for node A and C.

To help developing sensors easily in SCALEA-G, we provide sensor templates in C/C++ and Java. A sensor is developed based on the sensor template. All sensors are associated with some common properties such as sensor identifier, data schema, parameters and they interact with Sensor Manager Services by exchanging XML messages.

3.1 System Sensors and Sensor Repository

SCALEA-G provides a set of system sensors for monitoring the most commonly needed types of performance information on the Grid investigated by GGF DAMED-WG [11] and NMWG [15]. A system sensor may not really perform the measurement and monitoring; it can just query or collect data from others that actually perform the measurement.

Usually, system sensors are shared by various clients. To simplify the management and deployment of system sensors, a *sensor repository* is used to hold information about available system sensors such as properties of the sensor and parameters needed to invoke the sensor, etc. Each sensor repository is managed by a Sensor Manager Service that makes sensors in the repository available for use when requested.

[1] In our view, we classify Grid services into: computational services (e.g. host), software services (e.g. applications, middleware), and network services (e.g. connectivity, link). Basically, software services run on computational services and they can communicate with each other through network services.

```
<xsd:element name="sensorrepository" type="SensorEntry" />

<xsd:complexType name="SensorEntry">
  <xsd:sequence>
   <xsd:element name="desc" type="xsd:string"/>
   <xsd:element name="measureclass" type="xsd:string"/>
   <xsd:element name="schemafile" type="xsd:string"/>
   <xsd:element name="params" type="Params"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" />
  <xsd:attribute name="share" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="Params">
 <xsd:sequence>
    <xsd:element name="param" minOccurs="0"
                            maxOccurs="unbounded" >
      <xsd:complexType>
         <xsd:attribute name="name"   type="xsd:string"/>
         <xsd:attribute name="desc"   type="xsd:string"/>
         <xsd:attribute name="dataType" type="xsd:string"/>
      </xsd:complexType>
    </xsd:element>
 </xsd:sequence>
</xsd:complexType>
```

Figure 2. XML schema used to describe sensors in the sensor repository.

Figure 2 presents XML schema used to express sensors in the sensor repository. The XML schema allows to specify sensor-related information such as *name* (a unique name of the sensor), *measureclass* (implementation class), *schemafile* (XML schema of data produced by the sensor), *params* (parameters required when invoking the sensor), etc. For instance, a sensor namely *host.mem.used* which is used to measure ratio of used memory of a host is represented in the sensor repository as follows:

```
<sensor name="host.mem.used" share="yes">
    <desc>Ratio Used Memory</desc>
    <measureclass>scaleag.sm.sensors.Mem</measureclass>
    <schemafile>memusage.xml</schemafile>
    <params>
      <param name="Interval" desc="second" dataType="int"/>
    </params>
</sensor>
```

The corresponding XML schema of data provided by this sensor is shown in Figure 3. The *params* tag specifies an input parameter namely *Interval*, the period of time between two successive executions, required when invoking this sensor.

A client (user, service) makes requests *create sensor instance* by invoking a service operation on Sensor Manager Services or directly executing system sensor to create a new instance of a system sensor. By default, the lifetime of an instance will optionally be specified when the sensor instance is created.

3.2 Application Sensors

Application sensors are embedded in user's programs via source code instrumentation prior to application execution or via dynamic instrumentation at the application runtime.

```
<xsd:element name="sensordata" type="SensorData" />

<xsd:complexType name="SensorData">
  <xsd:sequence>
   <xsd:element name="hostname" type="xsd:string"/>
   <xsd:element name="eventtime" type="xsd:dateTime"/>
   <xsd:element name="availmem" type="xsd:double"/>
   <xsd:element name="usedmem" type="xsd:double"/>
  </xsd:sequence>
  <xsd:attribute name="name"
          type="xsd:NMTOKEN" fixed="host.mem.used"/>
</xsd:complexType>
```

Figure 3. XML schema of host.mem.used in sensor repository.

Application sensors are further classified into *profiling* and *tracing*. The first type of sensors supports profiling of user-selective performance metrics (e.g. timing, hardware parameters) of code regions in applications whereas the latter provides tracing of specific events at selected points in programs. Application sensor instances are automatically invoked when the application control flow reaches the relevant sensor code.

```
<xsd:element name="sensordata"   type="SensorData"/>
<xsd:complexType name="SensorData">
  <xsd:sequence>
   <xsd:element name="experiment" type="xsd:string"/>
   <xsd:element name="coderegion" type="CodeRegion"/>
   <xsd:element name="events"     type="Events"/>
   <xsd:element name="metrics"    type="Metrics"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:NMTOKEN"/>
</xsd:complexType>
```

Figure 4. Top-level XML schema of data provided by application sensors.

Figure 4 shows the top-level XML schema for data provided by application sensors. The *name* tag specifies kind of sensors, either *app.trace* or *app.prof* corresponding to tracing or profiling data, respectively. The *experiment* tag specifies a unique identifier determining the experiment (a sequential or parallel execution of a program on a given target architecture). This identifier is used to distinguish data between different experiments. The *coderegion* tag refers to information of the source of the code region (e.g. line, column). The *processingunit* tag describes the context in which the code region is executed; the context includes information about *grid site, computational node, process, thread*. The *events* tag specifies list of *events*, each of which consists of information about the time when the event occurs (*eventtime*), the name of the event (*eventname*) and a set of event attributes; an event attribute is represented in a tuple of attribute name and value (*attrname*=*attrvalue*). The *metrics* tag specifies a list of performance metrics each of which is represented in a tuple of name and value (*met-*

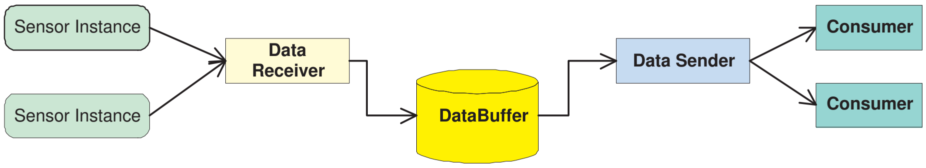

Figure 5. Data Service in Sensor Manager Service

ricname=metricvalue).

3.3 Sensor Manager Service

Sensor Manager Service consists of following components: *Service Administration, Data Subscription and Query, Data Publication, Instrumentation Request Forwarder* and *Data Service*.

Service Administration receives administrative requests and controls activities of the Sensor Manager Service and sensors in the Sensor Repository. Such activities are to activate a sensor (thus making a new sensor instance), to deactivate an existing sensor instance , to manage the data service. All activities in this component are carried out through invocations of Grid service operations.

Data Subscription and Query is responsible for processing subscribed/unsubscribed requests, queries for data from consumers. Based on the information published in the Directory Service, consumers can subscribe data provided by sensor instances. Resulting data will be *pushed* to consumers by the *Data Service* of Sensor Manager Service via the data channel. When a consumer unsubscribes a subscribed data type, the data service will stop sending that data to the consumer. Subscribing and unsubscribing tasks are implemented as operations of services. In supporting *pull* model, this component will process queries containing criteria of interesting data and send resulting data met the requested condition to the requester. Data query requests are represented in XPath/XQuery based on XML data schema published by sensor instances.

Data Publication implements tasks (adding, updating, removing) for registering information about Sensor Manager Service, sensor instances and properties of data produced by sensor instances with the Directory Service.

Instrumentation Request Forwarder receives instrumentation requests from clients and forwards these requests to corresponding mutator services which in turn conduct dynamic instrumentation.

Data Service receives data collected by sensor instances and delivers requested data to consumers. It implements filtering, searching, forwarding and caching data to/from various destinations/sources. When a consumer subscribes or queries data, the Sensor Manager Service will return the result to the consumer via a data channel. The data service has only one connection to each consumer for delivering multiple types of data. In the Data Service, shown in Figure 5, a *data receiver* is used to receive data from sensors and to store the received data into data buffers. The data receiver is a thread binding a well-known port and interacts with sensors via TCP connections. When a consumer subscribes/queries the data, a *data sender* will retrieve data from corresponding data buffers, filter the data and send the results to the requester.

3.4 Interactions between Sensors and Sensor Manager Services

The interactions between sensors and Sensor Manager Services involve the exchange of three XML messages.

In *initialization phase*, the sensor instance sends a *sensorinit* XML message which contains *sensor name*, an *XML Schema* of data sensor instance produces, *lifetime* in which the sensor instance is active and *description* information about the sensor instance to the Sensor Manager Service which then makes these information available for consumers via directory service.

In *measurement phase*, the sensor instance repeatedly performs measurement, encapsulates its measurement data into a *sensordataentry* XML message, and sends the message to the Sensor Manager Service. The measurement data will be enclosed by *<![CDATA[...]]>* tag. By doing so, sensors can customize the structure of their collected data. At the Sensor Manager Service side, the measurement data will be extracted and stored in appropriate data buffer.

Before the sensor instance finishes its execution, it sends a *sensorfinal* XML message to notify the Sensor Manager Service that it finishes sending its collected data to the Sensor Manager Service.

4 SCALEA-G Client Service

SCALEA-G Client Service provides facilities for other services to access SCALEA-G system. It consists of *Consumer Service*, *Performance Analyzer*, *Overhead Analyzer* and *Instrumentation Mediator*.

Consumer Service provides base facilities for clients to control and access available data in SCALEA-G. Through the Consumer Service, clients can perform subscription and queries on available data in Sensor Manager Services and Archival Services. This component also filters data received from many sources and forwards filtered data to corresponding calling client which processes that data. In addition, Consumer Service is used to relay control tasks, e.g. activating or deactivating sensors in Sensor Manager Services, from client to other SCALEA-G services.

Performance Analyzer and Overhead Analyzer are used to analyze data collected by sensor instances. Performance Analyzer can be used to analyze both system and application data whereas the Overhead Analyzer which is an extension of SCALEA overhead analysis [26] examines only performance data of Grid applications.

Instrumentation Mediator glues clients with instrumentation services. It gets instrumentation requests from clients and sends these requests to corresponding Source Code Instrumentation Service or Sensor Manager Service. The latter will forward the requests to mutator service. In the next section we will discuss the instrumentation service.

5 Instrumentation Service

Instrumentation Service aims to help instrumenting Grid services/applications. We support two approaches: source code and dynamic instrumentation. In the first approach, the Instrumentation Mediator gets source files along with instrumentation controls from the client. And then it sends all inputs to Source Code Instrumentation Service (SCIS) which is based on SCALEA Instrumentation System [26]. The instrumented sources are then compiled and can be executed in the target machines. SCIS service allows to specify code regions of which performance metrics (e.g. timing, hardware parameter, temporal overhead) of interest should be monitored and measured.

With the source code level approach, however, it requires the availability of all the source files. In addition, instrumentation and measurement metrics could not be configured at the runtime of the application. To overcome these problems, we exploit the dynamic instrumentation mechanism based on Dyninst [6]. In order to support the dynamic instrumentation, we implement a *mutator service* which contains calls into the Dyninst API and the code that implements the runtime compiler and the utility routines to manipulate the application process. A mutator service is responsible for controlling the instrumentation of application process on the compute node where the process is running. We develop an XML-based instrumentation request language (IRL) to allow the client specifying code regions of which performance metrics should be determined and controlling the instrumentation process. When an instance of mutator service running, it notifies its existence to the Sensor Manager Service and waits for control from the client. The client will be notified the existence of mutator services by subscribing information in directory service. The client then controls the instrumentation by sending IRL requests to mutator services which in turn will perform the instrumentation by inserting *snippets* (instrumentation code which implements application sensors) to application processes at program's selected points. Sensor instances will send data collected to Sensor Manager Services which in turn deliver that data to consumers.

5.1 Instrumentation Request Language (IRL)

The IRL is provided in order to facilitate the interaction between instrumentation requester (e.g. consumers, users, tools) and instrumentation service. IRL is an XML-based language which consists of instrumentation messages: request and response. Clients send requests to instrumentation services and receive instrumentation responses that describe the status of requests.

Figure 6 presents the XML data schema of IRL. The job to be instrumented is specified via *experiment* tag. Current implementation of IRL supports four requests including *attach, getsir, instrument, finalize*:

- *attach*: requests the instrumentation service to attach the application and to prepare to perform other tasks on that application.
- *getsir*: requests the dynamic instrumentation service to return SIR (Standardized Intermediate Representation) [12] of a given application.
- *instrument*: specifies code regions (based on SIR) and performance metrics should be instrumented and measured.
- *finalize*: notifies the instrumentation service that client will not perform any request on the given application.

In responding to a request from a client, the instrumentation service will reply to the client by sending an instrumentation response which contains the name of the request, the status of the request (e.g OK, FAIL) and possibly a detailed responding information which is encoded in *<![CDATA[...]]>* tag.

5.2 Example of IRL

In the following example, we present a simple example of IRL:

```
<?xml version="1.0" ?>
<irl>
  <experiment>
     <applicationName>/home/truong/cpi</applicationName>
     <jobID>7536</jobID>
     <experimentID>CPI-4</experimentID>
  </experiment
<request name="attach" />
```

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           xmlns:irl="http://www.par.univie.ac.at/project/scaleag/irl"
     targetNamespace="http://www.par.univie.ac.at/project/scaleag/irl"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified"
>

 <xsd:annotation>
  <xsd:documentation xml:lang="en">
   Instrumentation Request Language (IRL) schema for SCALEA-G
   Version 0.1
   Copyright 2003, Hong-Linh Truong. All rights reserved.
  </xsd:documentation>
 </xsd:annotation>

<xsd:element name="irl"   type="IRL"/>
<xsd:complexType name="IRL">
  <xsd:sequence>
    <xsd:element name="experiment" type="Experiment" minOccurs="0" maxOccurs="1"  />
    <xsd:element name="request"  type="Request" minOccurs="0" maxOccurs="unbounded" />
    <xsd:element name="response" type="Response" minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Request">
  <xsd:sequence>
    <xsd:element name="experiment"  type="Experiment" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="task"   type="Task" minOccurs="0"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:NMTOKEN"/>
</xsd:complexType>

<xsd:complexType name="Experiment">
 <xsd:sequence>
  <xsd:element name="applicationName" type="xsd:string"/>
  <xsd:element name="jobID"           type="xsd:string"/>
  <xsd:element name="experimentID"    type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Task">
  <xsd:sequence>
    <xsd:element name="coderegion" type="CodeRegion" maxOccurs="unbounded" />
    <xsd:element name="metrics" type="ListString"  />
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CodeRegion">
              <xsd:attribute name="name"   type="xsd:string"/>
              <xsd:attribute name="id"  type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="Response">
   <xsd:sequence>
    <xsd:element name="detail" type="xsd:string"/>
   </xsd:sequence>
   <xsd:attribute name="name" type="xsd:NMTOKEN"/>
   <xsd:attribute name="status" type="xsd:NMTOKEN"/>
</xsd:complexType>

<xsd:simpleType name="ListString">
  <xsd:list itemType="xsd:string"/>
</xsd:simpleType>

</xsd:schema>
```

Figure 6. XML Schema of Instrumentation Request Language

```
<request name="instrument">
   <task>
     <coderegion name="MPI_Reduce" />
     <coderegion name="MPI_BCast" />
     <metrics>WTIME L2_TCM L2_TCA </metrics>
  </task>
</request>
</irl>
```

The *experiment* tag specifies the information about the application such as application name, job identifier, etc. This information will be applied for all subsequent requests because there is no other experiment information stated inside any *request* tag. The first request *attach* asks the instrumentation service to attach the application process. The second request *instrument* instructs the instrumentation service to instrument all code regions *MPI_Reduce, MPI_Bcast* with performance metrics $WTIME$ (wallclock time), $L2\_TCA$ (level 2 cache accesses) and $L2\_TCM$ (level 2 cache misses).

6 Data Delivery, Caching and Filtering

6.1 Data Delivery

In SCALEA-G, messages containing collected data are propagating between various components. Data message is delivered in a separate data channel via secure sockets based on GSI and PKI. Message propagation uses a simply tunnel protocol as presented in Figure 7.

In this protocol, each sensor instance builds its XML data message and sends the message to a Sensor Manager Service. When a Sensor Manager Service receives a data message from a sensor instance, it already knows the sensor sending the message (from initialization step) so that it stores the message into an appropriate buffer. When a client subscribes and/or queries data (by invoking corresponding operations of consumer service), the Sensor Manager Service will return a *ResultID* to the consumer service. The Sensor Manager Service then builds XML messages by tagging the ResultID to the data met the subscribed/queried condition and sends these messages to the consumer service via the data channel. At the consumer service side, the messages will be filtered based on ResultID. The payload of the messages will then forward to the calling client.

6.2 Caching and Filtering Data

Data produced by system sensors will be cached in ring buffers at Sensor Manager Service. Therefore, data provided by one system sensor instance can be shared among multiple consumers. In the current implementation, for each type of system sensor, a separate data buffer is allocated for holding data produced by all instances of that type of sensor. The size of the data buffer can be controlled by the administrator. When a data buffer is full, the obsolete data will be removed in order to free space for the new data entry. In *push* model (consumers subscribe data), any new data entry met the subscribed condition will always be sent to existing subscribed consumers. In *pull* model, Sensor Manager Service searches current available entries in the data buffer and only entries met conditions of consumer query will be returned to the requested consumers.

Buffering data produced by application sensors is similar to that for system sensors. However, we assume that there is only one client to perform the monitoring and analysis for each application and we do not limit the size of the data buffer.

7 Security Issues

SCALEA-G requires the authentication and authorization in several actions including registrations and queries to the directory service, subscriptions to Sensor Manager Services to receive data, controls activities of Sensor Manager Services and their managed sensors. The authentication and authorization are carried out by using GSI provided by Globus [14]. Stream data delivery is performed via secure sockets based on GSI and PKI whereas Grid service operations use transport-level security.

For few system sensors, information collected has to be delivered to the right consumer, for example, collected information about malicious attacks on a Grid site should only be accessed by site administrator. Therefore, SCALEA-G services impose controls on clients in accessing SCALEA-G services and data. In the case SCALEA-G services are started by the administrator for multi-user, each SCALEA-G service (e.g. Directory Service, Archival Service, Sensor Manager Service) contains an Access Control List (ACL) which maps client's information to sensors the client can access. The ACL can statically be set up by the administrator. The client information obtained from client's certificate when the certificate is used in authentication will be compared with entries in the ACL in the authorization process.

In the case of monitoring and analyzing application, subscribing and/or querying data provided by application sensors, client's information will be recorded. Similarly, before application sensor instances start sending data to the Sensor Manager Service, the Sensor Manager Service obtains information about the client who executed the application (in initialization step). Both sources of information will be used for authorizing the client in receiving data produced by application sensors.

8 Implementation Overview

We are currently implementing the prototype of our design. Most code is written in Java except that part of sensor

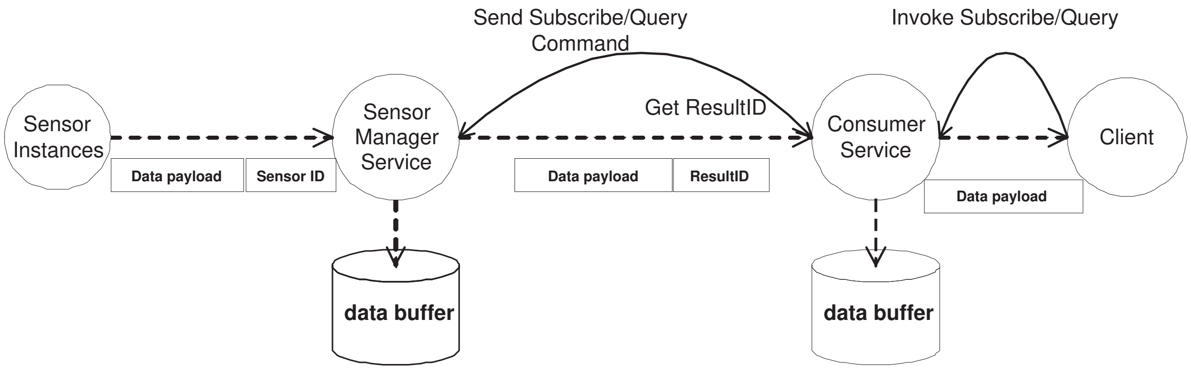

Figure 7. Data Delivery, Caching and Filtering

library and mutator service are written in C/C++. Globus Toolkit [17] and JavaCog [19] are used as base libraries for implementing OGSA-enabled services, communication and security in SCALEA-G services.

The mutator service uses Globus C library for communication and security tasks and the facilities for dynamic instrumentation of applications at runtime are provided by Dyninst [6]. We have implemented a library for processing IRL based on Xerces C++ library [28].

PostgreSQL [23] has been used for archiving information in the directory service. To buffer the data at Sensor Manager Service and Client Service, we use Berkeley DB provided by Sleepycat [9].

9 Experiments and Examples

In this section we present few experiments and examples of SCALEA-G. Experiments are conducted via SCALEA-G User Portal.

9.1 Administrating Sensor Manager Services and Sensors

Figure 8 presents the administration portal which is used to manage activities of Sensor Manager Services. By selecting a Sensor Manager Service, a list of available sensors and a list of sensor instances (active sensors) managed by that Sensor Manager Service will be presented in the top-left and top-right window of Fig. 8, respectively. An administrator can make a request creating a new sensor instance by selecting a sensor and clicking the *Activate* button. When creating an sensor instance, lifetime and parameters will be specified, e.g. Figure 8 shows the dialog for setting input parameters for *path.delay.roundtrip* sensor based on its data in the sensor repository. Similarly, an existing sensor instance can be deactivated (by selecting *Deactivate* button). In addition, the administrator can control (*Add, Update, Remove*) the registration of information about Sensor Manager Services, sensor instances and properties of data they provide with the directory service (see *Registered Sensor* window in Fig. 8). By choosing a sensor, detailed information of that sensor (e.g. parameters, XML schema) will be shown in the two bottom windows.

The administrator can also activate or deactivate the data receiver in a Sensor Manager Service. If the data receiver is disable, the Sensor Manager Service will not receive data sent by sensor instances. When activating the data receiver, the administrator can specify the port on which the data receiver is binding.

9.2 Dynamic Instrumentation Example

Figure 9 depicts the portal for interacting with the dynamic instrumentation in SCALEA-G. On the top-left window, the user can choose a directory service and retrieve a list of instances of mutator service registered to that directory service. The user can monitor processes running on compute nodes where instances of mutator service execute by invoking *Get/Update User Processes* operation (or can be done automatically via subscription), e.g. active processes in node *bridge.vcpc.univie.ac.at* are shown in the top-right window of Figure 9. For a given application process, its SIR (currently only at level of program unit and function call) can be obtained via *Get SIR* operation, e.g. the SIR of *rpp*3*do* process is visualized in the bottom-right window in Figure 9. In the bottom-left window of Figure 9, the user can edit IRL requests and send these requests to selected instances of mutator service, asking them to perform the instrumentation of selected applications.

9.3 Performance Monitoring and Analysis User Portal

This portal allows the user to monitor and analyze online performance behavior of Grid systems and applications at

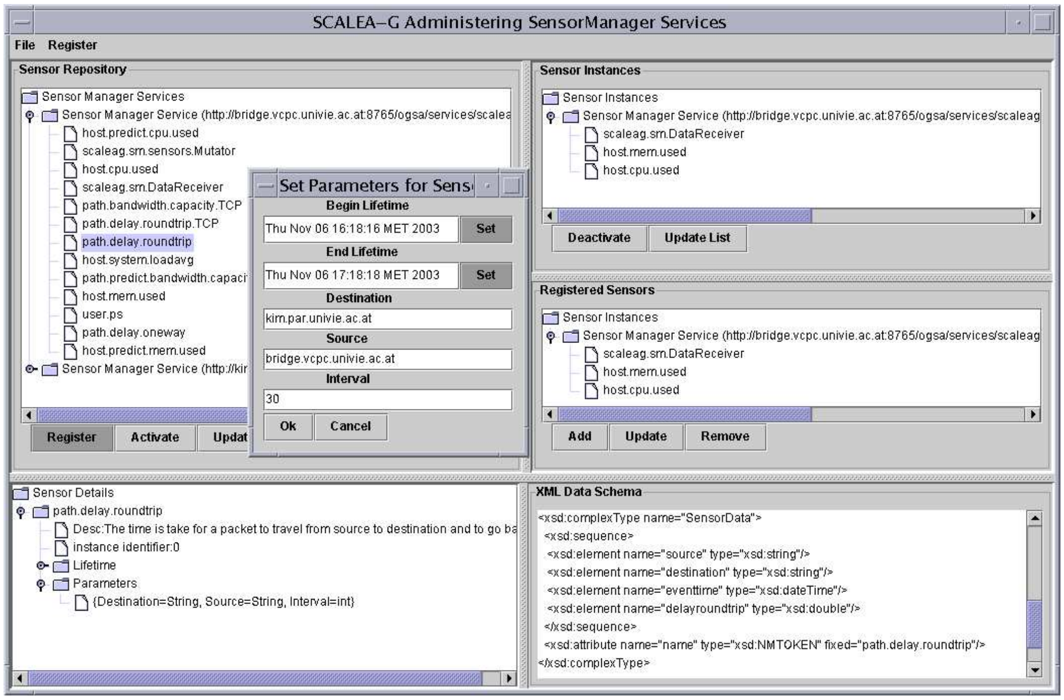

Figure 8. SCALEA-G Administration Portal

the same time. Performance data can be obtained via push or pull model:

- *Push model*: The user just subscribes data that sensor instances provide by choosing appropriate available sensor instances. Whenever a new data sent to a Sensor Manager Service, the Sensor Manager Service will push the new data to its existing subscribed consumers.

- *Pull model*: Based on published XML schema data, the user edits XPath/XQuery-based queries and then sends these queries to corresponding Sensor Manager Services which in turn will search their data buffers and send searching results to client.

For each sensor instance, the user can examine data properties of the sensor instance by clicking into the name of the instance in the list of active sensors. For instance, in Figure 10, an instance of *host.system.loadavg* sensor is examined and its XML schema is shown in bottom-right window. The user can subscribe or query data by choosing *Subscribe* or *Query* (see Figure 10). For example, we subscribed application data of experiment *3DPIC-4N-3* and data provided by *host.mem.used* sensor and then examined that data online (see *Application Online Analysis* and *Monitoring Memory Usage* window in Figure 10). When the user does not want to receive subscribed data, he can cancel the subscription by selecting the sensor instance in the list of subscribed sensors and clicking the *Unsubscribe* button.

For system monitoring, we have provided a set of system sensors for monitoring Grid hosts and networks, some described in Table 1.

For applications, both event and profile analysis are supported. For example, profile analysis provides functions for analyzing subscribed or queried data including overhead analysis, load imbalance, metric ratio, etc.

10 Related Work

To assist developers in building monitoring tools for Grid system, Global Grid Forum (GGF) [16] has proposed Grid Monitoring Architecture (GMA) [1] model which describes the major components of a Grid monitoring tool and their essential interactions. GMA has been adopted by many Grid monitoring tool developers. For example, Relational GMA (R-GMA) [7], developed in the DataGrid, exposes a relational model with SQL support to provide static as well as dynamic information about Grid resources.

Several existing tools are available for monitoring Grid computing resources and networks. For instance, MDS (a.k.a GRIS) [8] provided by Globus [17] is based on hierarchy model in which an index service is used to man-

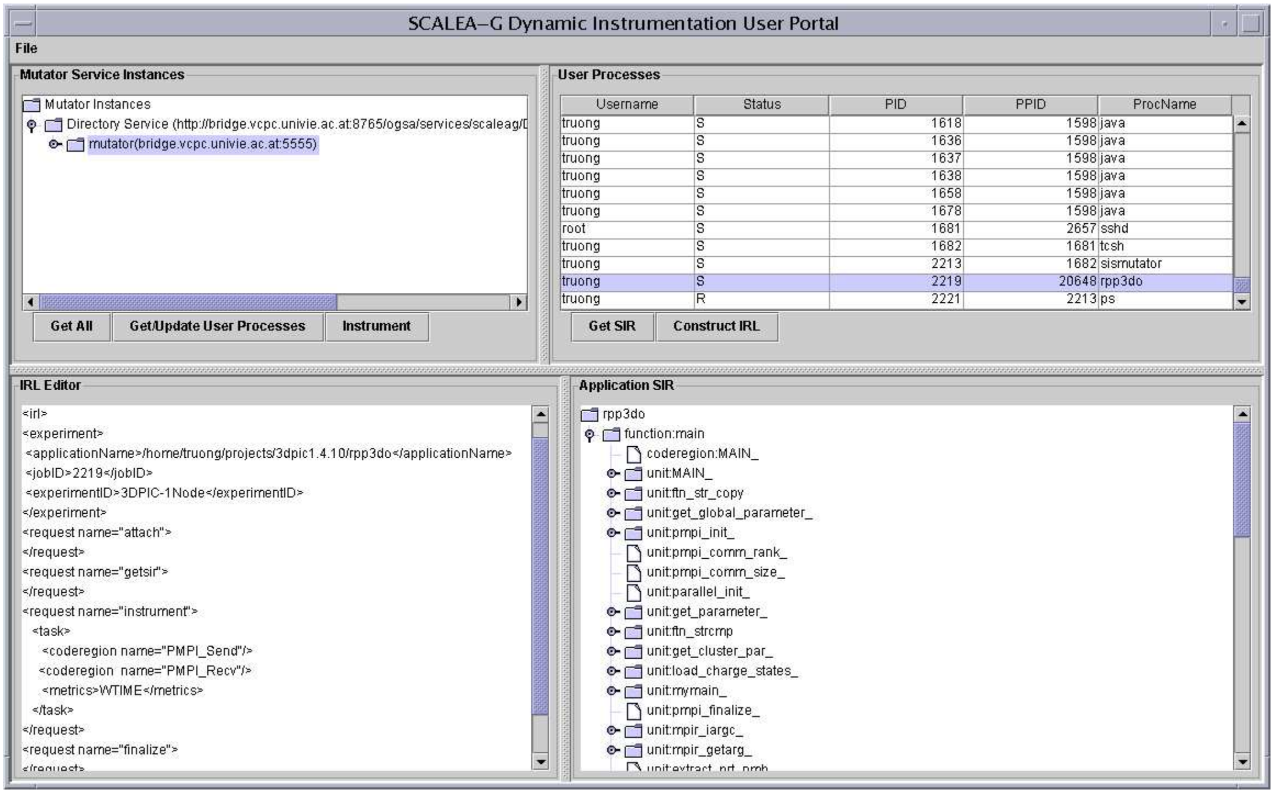

Figure 9. SCALEA-G Dynamic Instrumentation Portal

age monitoring services for resources. The latest version of MDS provides both static and dynamic data about computational resources and job submission information in XML format. Network Weather Service (NWS) [27] provides a set of system sensors such as CPU, network sensors that periodically monitors and dynamically forecasts the performance of various networks and computational resources. Mark Baker and Garry Smith present a prototype Grid-site Monitoring System (GridRM) [2, 3] which is based on three-tier architecture. GridRM accesses information of resources from other services such as MDS, SNMP and uses relational database for storing information.

In contrast to the number of system monitoring tools, few monitoring and performance analysis tools for Grid applications have been introduced. For instance, NetLogger [24, 18] provides tracing functionalities for distributed applications. However, performance analysis is carried out in post-mortem fashion. GRM [21] is a semi-on-line monitor that collects information about an application running in a distributed heterogeneous system and delivers the collected information to the PROVE visualization. The application is manually instrumented by inserting C functions. On the host where the application processes are running, a local monitor is used to handle trace data recorded by application sensors and stored in shared memory buffer. The local monitor will send trace data to the main monitor when the shared memory buffer is full. Recently GRM has been interfaced to R-GMA in order to collect trace information about message passing parallel programs in Grid [22]. In both GRM and NetLogger, however, the instrumentation has to be done manually.

OCM-G [4] is an infrastructure for Grid application monitoring which uses OMIS [20] as a communication interface. Application monitor embedded in the monitored application collects performance data and passes the data to a local monitor which resides on each node. At the top of OCM-G, G-PM [5] which is targeted to interactive Grid application is used to conduct the performance analysis. Although OCM-G/G-PM support dynamic instrumentation, they however require the user to link the application with MPI wrapper libraries consisting of probes used to collect performance data. Based on that, G-PM then controls conditional variables to activate/deactivate instrumentation code. Therefore, currently the dynamic instrumentation is

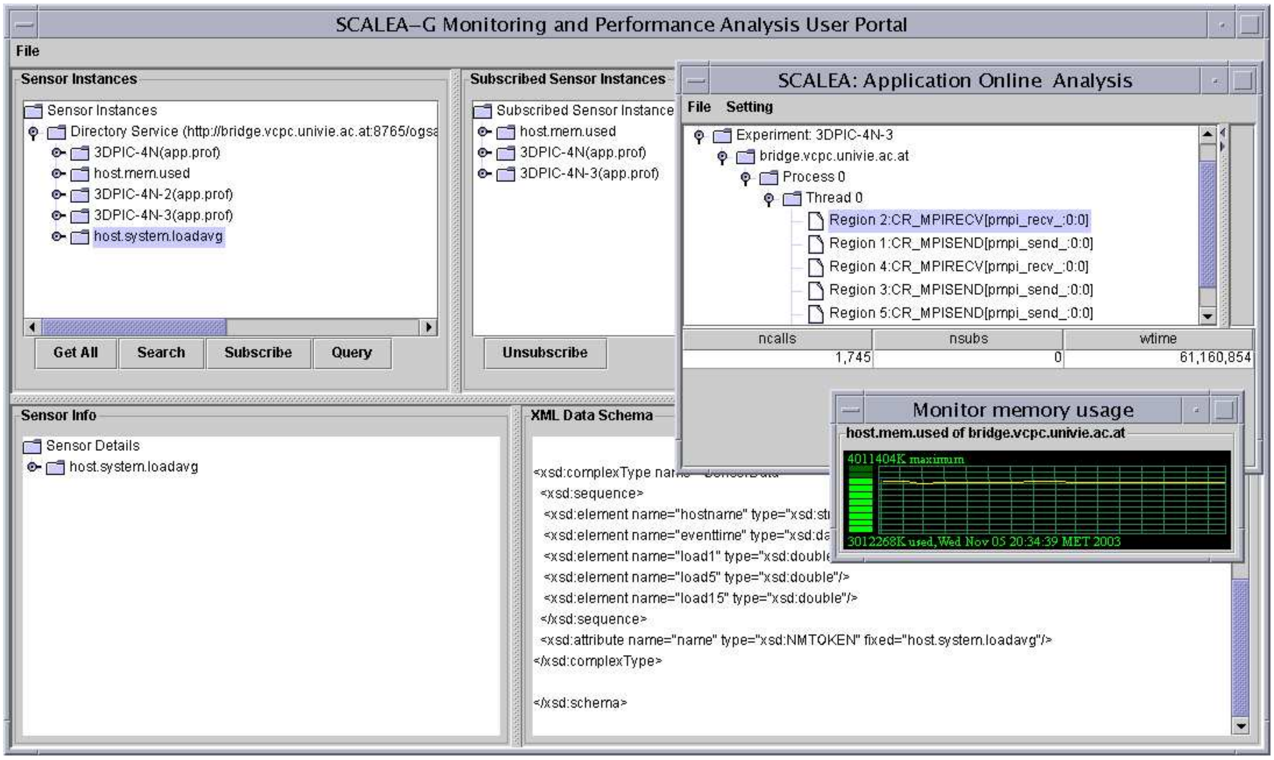

Figure 10. Performance Monitoring and Analysis Portal

limited to pre-instrumented MPI functions.

None of aforementioned systems, except MDS, is OGSA-based Grid service. Consequently, they do not well address challenges in Grid environment such as service discovery, dynamic service creation, lifetime management. It is rather difficult to deploy these tools in OGSI framework. Differing from most aforementioned and existing Grid monitoring tools, SCALEA-G is based on OGSA and supports both Grid infrastructure and application monitoring and performance analysis in a unified system, thus increasing the chance to relate measurement metrics of various sources at different levels.

There are well-known tools supporting dynamic instrumentation, e.g. Paradyn [25], DPCL [10], however, these tools are designed for conventional parallel systems rather than Grids; to date we are not aware of any result of using these tools in Grid. In addition, the requests for conducting dynamic instrumentation are built based on a specific protocol. The lack of a well-defined, general purpose protocol like IRL has hindered other services from using these tools to conduct the dynamic instrumentation. Our work on developing IRL and general-purposed instrumentation service intendedly provides a common means for a large number of other services to conduct dynamic instrumentation of interesting applications while hiding all low level details of instrumentation mechanism.

Furthermore, existing tools employ a non-widely representation of performance data, lack published schema of data they provide. That prevents other services to easily access their data. SCALEA-G, in contrast, uses widely-accepted XML format for representing performance data, publishes all data schema, and providing query mechanism with XPath/XQuery-based requests. As a result, other services will smoothly access the data.

11 Conclusion and Future Work

In this paper we presented the architecture of SCALEA-G which is a unified monitoring and performance analysis system in the Grid. We have described main SCALEA-G services in details. SCALEA-G, based on OGSA combined GMA concept, provides a set of Grid services for monitoring and conducting performance analysis in Grid systems. The main contributions of this paper center on 1) the unique monitoring and performance analysis system based on open grid service architecture (OGSA) and 2) an instrumentation request language (IRL) in accompany with a generic instru-

Sensor Name	Monitored Object	Description
host.cpu.used	host	The average, over the last minute, of the amount of time that processors of this system were not idle
host.predict.cpu.used	host	The prediction of CPU usage.
host.mem.used	host	The ratio used memory.
host.predict.mem.used	host	The prediction of ratio used memory.
host.system.loadavg	host	The average number of jobs in the run queue over the last 1, 5 and 15 minutes.
path.bandwidth.capacity.TCP	network	The maximum amounts of MB/s can be transfered from source to destination using TCP.
path.predict.bandwidth.capacity.TCP	network	prediction of the bandwidth capacity using TCP protocol.
path.delay.oneway	network	The time is taken for a packet to travel from source to destination
path.delay.roundtrip	network	The time is taken for a packet to travel from source to destination and to go back.
path.delay.roundtrip.TCP	network	The time is taken for a packet to travel from source to destination and to go back using TCP protocol.

Table 1. SCALEA-G's system sensors

mentation service in Grid.

Yet, there are many rooms for improving the system. Firstly, the set of sensors will be extended, enabling to monitor more resources and services, and providing more diverse kinds of data. The language IRL will be extended to allow specifying more complex instrumentation requests such as events. Moreover, IRL currently is used in communication between clients and dynamic instrumentation services. We intend to support IRL in our source code instrumentation service as well.

We are investigating to enhance our SCALEA multi-experiment analysis to support the performance comparison of several experiments in Grid. With the proliferating number of Grid services being implemented in Java, we are considering to develop an instrumentation service for Java-based services.

References

[1] B. Tierney et. al. A Grid Monitoring Architecture. Technical report, Performance Working Group, Grid Forum, January 2002. http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-2.pdf.

[2] Mark Baker and Garry Smith. A Prototype Grid-site Monitoring System. Technical report, The Distributed Systems Group, University of Portsmouth, January 2002. Version 1.

[3] Mark Baker and Garry Smith. GridRM: A resource monitoring architecture for the Grid. *Lecture Notes in Computer Science*, 2536:268–??, 2002.

[4] Bartosz Balis, Marian Bubak, Włodzimierz Funika, Tomasz Szepieniec, and Roland Wismüller. An infrastructure for Grid application monitoring. *Lecture Notes in Computer Science*, 2474:41–??, 2002.

[5] Marian Bubak, Włodzimierz Funika, and Roland Wismüller. The CrossGrid performance analysis tool for interactive Grid applications. *Lecture Notes in Computer Science*, 2474:50–??, 2002.

[6] Bryan Buck and Jeffrey K. Hollingsworth. An API for Runtime Code Patching. *The International Journal of High Performance Computing Applications*, 14(4):317–329, Winter 2000.

[7] Rob Byrom, Brian Coghlan, Andrew W Cooke, Roney Cordenonsi, Linda Cornwall, Abdeslem Djaoui, Laurence Field, Steve Fisher, Steve Hicks, Stuart Kenny, Jason Leake, James Magowan, Werner Nutt, David O'Callaghan, Norbert Podhorszki, John Ryan, Manish Soni, Paul Taylor, and Antony J Wilson. Relational grid monitoring architecture (R-GMA). August 15 2003. Comment: Talk given at UK e-Science All-Hands meeting, Nottingham, UK, September 2-4, 2003. 7 pages of LaTeX and 5 PNG figures.

[8] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the Tenth IEEE*

International Symposium on High-Performance Distributed Computing (HPDC-10). IEEE Press, August 2001.

[9] Sleepcat Berkeley db project. *http://www.sleepycat.com*.

[10] L. DeRose, T. Hoover Jr., and J. Hollingsworth. The dynamic probe class library: An infrastucture for developing instrumentation for performance tools. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS-01)*, pages 66–66, Los Alamitos, CA, April 23–27 2001. IEEE Computer Society.

[11] Discovery and Monitoring Event Description (DAMED) Working Group. http://www-didc.lbl.gov/damed/.

[12] T. Fahringer, M. Gerndt, Bernd Mohr, Martin Schulz, Clovis Seragiotto, and Hong-Linh Truong. Standardized Intermediate Representation for Fortran, Java, C and C++ Programs. APART Working group (http://www.kfa-juelich.de/apart/), Under development, June 2003.

[13] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid Services for Distributed System Integration. *IEEE Computer*, pages 37–46, June 2002.

[14] Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. In *Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS-98)*, pages 83–92, New York, November 3–5 1998. ACM Press.

[15] GGF Network Measurements Working Group. http://forge.gridforum.org/projects/nm-wg/.

[16] Global Grid Forum. http://www.gridforum.org/.

[17] Globus Project. http://www.globus.org.

[18] D. Gunter, B. Tierney, B. Crowley, M. Holding, and J. Lee. Netlogger: A toolkit for distributed system performance analysis. In *Proceedings of the IEEE Mascots 2000 Conference*, August 2000.

[19] G. Laszewski, I. Foster, J. Gawor, and P. Lane. A java commodity grid kit. *Concurrency and Computation: Practice and Experience*, 13(643-662), 2001.

[20] T. Ludwig and R. Wismueller. OMIS 2.0 — A universal interface for monitoring systems. *Lecture Notes in Computer Science*, 1332:267–276, 1997.

[21] N. Podhorszki and P. Kacsuk. Design and implementation of a distributed monitor for semi-on-line monitoring of visualmp applications. In *DAPSYS'2000 Distributed and Parallel System, From Instruction Parallelism to Cluster Computing*, pages 23–32, Balatonfured,Hungary, 2000.

[22] N. Podhorszki and P. Kacsuk. Monitoring Message Passing Applications in the Grid with GRM and R-GMA. In *Proceedings of EuroPVM/MPI'2003*, Venice, Italy, 2003.

[23] PostgreSQL 7.1.2. http://www.postgresql.org/docs/.

[24] B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks, and D. Gunter. The netlogger methodology for high performance distributed systems performance analysis. In *Proceeding of IEEE High Performance Distributed Computing conference*, July 1998.

[25] Paradyn Parallel Performance Tools. http://www.cs.wisc.edu/paradyn/.

[26] Hong-Linh Truong and Thomas Fahringer. SCALEA: A Performance Analysis Tool for Parallel Programs. *Concurrency and Computation: Practice and Experience*, 15(11-12):1001–1025, 2003.

[27] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computing Systems*, 1999.

[28] Xerces C++ Parser. http://xml.apache.org/xerces-c/. Apache XML Project.

[29] XML Path Language. http://www.w3.org/tr/xpath.html.

[30] XML Schema. http://www.w3.org/xml/schema.

[31] XQuery: an XML Query Language. http://www.w3.org/tr/xquery/.