Scientific Programming 15 (2007) 173–188
IOS Press

VLAM-G: Interactive data driven workflow engine for Grid-enabled resources

Vladimir Korkhov*, Dmitry Vasyunin, Adianto Wibisono, Adam S.Z. Belloum, Márcia A. Inda, Marco Roos, Timo M. Breit and L.O. Hertzberger
Faculty of Science, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
E-mail: {vkorkhov, dvasunin, wibisono, adam, inda, roos, breit, bob}@science.uva.nl

Abstract. Grid brings the power of many computers to scientists. However, the development of Grid-enabled applications requires knowledge about Grid infrastructure and low-level API to Grid services. In turn, workflow management systems provide a high-level environment for rapid prototyping of experimental computing systems. Coupling Grid and workflow paradigms is important for the scientific community: it makes the power of the Grid easily available to the end user. The paradigm of data driven workflow execution is one of the ways to enable distributed workflow on the Grid. The work presented in this paper is carried out in the context of the Virtual Laboratory for e-Science project. We present the VLAM-G workflow management system and its core component: the Run-Time System (RTS). The RTS is a dataflow driven workflow engine which utilizes Grid resources, hiding the complexity of the Grid from a scientist. Special attention is paid to the concept of dataflow and direct data streaming between distributed workflow components. We present the architecture and components of the RTS, describe the features of VLAM-G workflow execution, and evaluate the system by performance measurements and a real life use case.

Keywords: Grid, data driven workflow engine, direct data streaming

1. Introduction

Grids have emerged as a global cyber-infrastructure for the next-generation e-Science applications. Scientific communities utilize Grids to share, manage and process large data sets. Complex scientific experiments often require access to distributed resources such as computational resources, data repositories, third-party applications, and scientific instruments. The utilization of these resources requires multi-domain expertise which is beyond the common knowledge of a single scientist. Scientific workflows are designed to automate scientific processes based on data dependencies and their control. They also aim at abstracting the usage of the necessary underlying resources in order to help the scientist to focus on his own research [16]. A workflow management system provides the means to compose, execute and monitor the workflow. The core component of a workflow management system is an engine responsible for the execution of a scientific workflow.

In the context of the VL-e [9] project we focus on the development of the VLAM-G workflow management system for the e-Science framework. A wide range of e-science applications from different scientific domains, namely high energy physics, bio-informatics, bio-diversity, bio-medicine, and tele-science [1] shows that data access and processing play an important role. A typical scenario involves a number of steps: (a) access data on a remote storage system; (b) move the data to one of the available computing nodes; (c) process the data, which results in a new data set (intermediate data) that is further processed on another node and thus needs to be moved again. These steps are repeated until the final data sets are generated and stored in a predefined location.

Frequently, when following this scenario, scientists would prefer to delegate all non-scientific activities

*Corresponding author: Kruislaan 403, 1098 SJ, Amsterdam, The Netherlands. Tel.: +31 205257599; Fax: +31 205257490; E-mail: vkorkhov@science.uva.nl.

ISSN 1058-9244/07/$17.00

such as finding the appropriate available resources and manipulating input, output, and intermediate data sets to a system they trust. Hiding such details of the underlying grid execution from the end user allows them to concentrate on the essential tasks they need to perform in order to achieve their scientific goals, e.g. defining original data sets, defining processes that need to be performed and the order in which these processes must be executed. On the other hand, the in-depth view at underlying processing on the low-levels can be provided as well, and an interested user can check what particular resources are used, examine intermediate results of execution and monitor actual computational processes. The ability to control one or a few parameters of the processes interactively, without having to stop or restart experiments, is also desirable.

In the scenario above, as well as in many other scenarios describing scientific experiments such as weather predictions [15], bio-medical visualization [12], knowledge discovery in databases [14,13], and other scientific domains [14], data is one of the main drivers of progress in these experiments. In the paper [31] the authors describe three different use cases from chemistry, cosmology and astronomy, where different patterns of pipelined processes have to be executed concurrently on distributed computing resources. For performance reasons data is streamed between the components composing the different levels of the pipeline.

As described in [32], processing data in streams has certain advantages because it can be done on the fly, reducing the need to store intermediate data which is not needed. Therefore, a data-driven workflow engine is the most appropriate to support efficient execution of this class of scientific experiments.

In this paper, we present a dataflow driven workflow engine, which uses the basic services of the Grid to allow data streams to be established efficiently and transparently between remote processes composing a scientific workflow. This workflow engine consists of a Run-Time Environment (*libvlport* library) for workflow components and a Run-Time System Manager (*RTSM*) used to control and orchestrate the execution of the entire workflow.

The experiment model used in the VLAM-G is a data-driven model, where scientific experiments can be composed as a workflow which executes its components on the available Grid resources. The engine orchestrating the execution of the workflow initiates 'point-to-point' data streams between workflow components allowing intermediate data to flow along the workflow pipeline. All the resources needed for such data stream driven distributed processing have to be available simultaneously in contrast to typical Grid usage scenario with resources joining and leaving. Thus a significant role is played by co-allocation stage that ensures simultaneous reservation of the resources needed. This stage is a complex and broad topic itself, and can be a theme for a separate paper; here we address the issues of execution and control of a data-driven workflow not considering co-allocation in details.

This paper is organized as follows. Section 2 reviews related work in the area of workflow management on the Grid. Section 3 presents a general view of the VLAM-G workflow management system, describing its architecture and the architecture of its subsystems. Section 4 describes the implementation of the runtime system library and the wrappers that allow developing workflow components using a number of programming languages. Section 5 shows performance evaluation results. Section 6 presents a real life example of a VLAM-G experiment from bioinformatics field. Bioinformatics is one of the key areas that has the requirements for data driven workflow and pipelined data processing which is illustrated in [44–46]. We conclude the paper with a discussion of future work in the context of the on-going VL-e project.

2. Related work

Within the e-Science and Grid communities, Workflow Management Systems (WMS) are the subject of intensive research since they provide an appropriate abstraction level to allow any scientist to take advantage of the capabilities of geographically distributed resources [39,40]. Different approaches have been proposed to formulate the workflow concept and to promote the development of diverse trends in terms of the functionality and features of workflow systems.

Grid-enabled WMS(s), especially in e-Science, often have to manage geographically distributed concurrent computational processes that exchange data in a peer to peer fashion across different security domains. To achieve this goal, different approaches have been developed by various research groups. For example, the P-GRADE portal [33] operates defining dependencies between different components with respect to their execution order which is appointed by output/input file transfers between jobs. The management of workflow in P-GRADE is performed by a WMS based on Condor DAGMan [34] with extensions to provide necessary file transfers.

The recent trend towards Service Oriented Architecture (SOA) stimulated the development of another category of WMS(s) targeting the composition of services, often implemented as Web services (e.g. Taverna [26] and Triana [27]). In the case of the Triana project, the proposed WMS supports the composition of workflows where components can be executed locally, as Web services, or accessed via the GAT-interface job submissions [28]. Taverna is another WMS popular within the bio-informatics community; it is mostly oriented to work with a set of biological web services, it has a number of additional components which allow semantic annotation of workflow components and data provenance.

In e-Science, there is a large set of libraries and applications that is difficult to re-implement to fit the new grid-computing paradigm; This software is considered as legacy code that cannot be modified. The manipulation of the legacy code is an important feature for both WMS supporting SOA based components and the ones utilizing job submissions to grid-enabled resources.

Another important feature of some e-Science applications is the demand for direct data streaming between distributed processes on the grid, especially for application domains that rely upon semi-realtime data processing. A number of workflow management systems focus on the development of robust and efficient data streaming over the Grid. Some of these systems focus on enabling data streaming capabilities, such as SCiFLo [18] where the data streaming is provided by binary channels, GriddLeS [19] where streaming functionality is supported over pipes, and Narada Brokering [20], which supports the interesting concept of hybrid streams where multiple "simple streams" are intrinsically linked. Some systems offer more complete frameworks enabling more control and steering of the streams. The UniGrids Streaming Framework [21] provides steering capabilities as well as data streaming. A UniGrids Web Service is used to control a set of available stream types, to create streams, and to manage already created ones. The Styx Grid Service (SGS) [22] is another example of a streaming framework which uses a remote service type that allows data to be streamed directly between service instances. The Styx clients can monitor progress and status through persistent connections. SGS can interoperate with other service types such as Web Services in a workflow. Other systems such as ASSIST [23] propose high-level structured parallel programming capabilities which includes a skeleton-based language, and a set of compiling tools and runtime libraries for supporting the data streaming. The DSP Grid [24] is conceptualized as a workflow engine with the capability of dynamically marshalling the required data stream resources, planning the decomposition of the process graph and dispatching the component work across Grid resources to maintain data stream processing in real-time.

To describe application workflows, a variety of workflow description languages are used. Most of the existing WMS use their own languages with different levels of complexity. For example, in Taverna data models can be represented using SCUFL (Simple Conceptual Unified Flow Language), they consist of inputs, outputs, processors, dataflow, and control flow. In addition to specifying execution order, the control flow can also be triggered by state transitions during the execution of parent processors. Askalon [25] employs AGWL (Abstract Grid Workflow Language) which provides a set of constructs to express sequence, parallelism, choice and iteration workflow structures. Teuta, another Askalon component, supports graphical specification of Grid workflow applications based on the UML activity diagram which is a graphical interface to AGWL. For a complete survey and taxonomy on existing workflow systems we refer the reader to [17].

VLAM-G provides a basic set of capabilities for building workflows by connecting components to each other based on data dependencies. As a core concept VLAM-G uses dataflow between simultaneously executing distributed components as an execution driving activity, while many other systems employ control flow and/or only perform sequential execution of workflow steps according to intermediate data readiness. Intermediate data handling is also performed differently. Some systems like Triana and Taverna use a centralized approach whereby all the data is controlled and collected at the central point where the engine resides. In turn, VLAM-G and P-GRADE transfer data directly between the participating components. Another distinguishing feature of VLAM-G is its ability to control an executing workflow at runtime in several ways. Firstly, the workflow components can be parameterized; the parameters can be controlled during execution. Secondly, as VLAM-G operates with remote jobs co-allocated at runtime, it enables direct access to the graphical output of a component (i.e. component GUI, if one exists) by providing a shared virtual display. To the best of our knowledge no other system provides similar capabilities.

3. A data-driven workflow engine

3.1. The vision

The aim of the VLAM-G system developed at the University of Amsterdam is to provide and support coordinated execution of distributed Grid-enabled components combined in a workflow. This system combines the ability to take advantage of the underlying Grid infrastructure and a flexible high level rapid prototyping environment. On the high level, a distributed application is composed as a data driven workflow where each component represents a process or service on the Grid. Processes are activated only when the data is available on their input ports. The significant difference from other similar systems is the support for simultaneous execution of co-allocated processes on the Grid which enables direct data streaming between the distributed components: traditional batch processing of grid jobs and workflow execution based on input/output files exchange between the components is not suitable for many use case scenarios. This feature is highly required for semi-realtime distributed applications e.g. in the bio-medical domain or in online video processing and analysis [30].

Grid technology is maturing very quickly, the new paradigm based on SOA is replacing monolithic architectures and the original resource oriented approach to Grid computing. However, the transition to the new paradigm will not take place overnight; moreover, the old approaches remain more efficient in a number of cases. One of the targets of VLAM-G is, thus, to enable support for co-existence of different types of grid execution models within a single workflow. This goal is achieved by abstracting a particular Grid execution model to an intermediate common representation. In VLAM-G a workflow is not composed of particular specific Grid jobs or services but of components with a special interface. These components are called modules, and they are the core entities of the VLAM-G data-driven workflow. Thus a module can represent a specially developed application which uses the VLAM-G native module API (libvlport), a web service, or a legacy application.

The runtime control of the execution of a distributed workflow provides the ability to monitor the execution and influences the behavior of workflow components. VLAM-G supports several ways of runtime control: direct interaction with the user interface of a module (remote X GUI access) and module parameter control (reading flags and values set by a module and updating these values from outside the module). Monitoring delivers all the log data from remote modules to the VLAM-G user interface thus all the issues in module execution can be tracked centrally.

Intensive distributed data processing might take a long time. To facilitate the handling of the executing workflow, the system is capable of closing the user interface, detaching from the workflow engine and reattaching later on during runtime.

The core features of the system we present are: (1) Dataflow (and not control flow) is used as a driving force; (2) Workflow components (VLAM-G modules) are versatile: a module may be either a specially developed software component (written in C++/Java/Python using the VLAM-G API), or an interface to legacy applications or web-services; (3) Distributed execution: support for Grid job submissions together with web serivces and local tasks within a single workflow; (4) Support for legacy applications wrapped as modules (flexible XML configuration); (5) Support for remote graphical output: remote X display for Grid jobs is provided; (6) Interactivity support: online control via parameters, and via remote graphical output; (7) Decentralized handling of intermediate data; (8) Decoupling of GUI and engine;

3.2. The architecture

In this section we present the architecture of the VLAM-G workflow engine which enables the execution and the runtime control of distributed data-driven workflows on the Grid. A workflow is composed of a set of components called modules which represent an executing entity (remote application or a web service). As computing resources VLAM-G can use: (1) grid enabled local or remote sites (so far VLAM-G works only with the Globus middleware [7]) that enable inbound and outbound communications; this allows data streams to be established between workflow components located on remote grid resources. A special case is the multi-cluster environment where it is not common that the internal worker nodes have public addresses and direct connectivity to the outside. This is a serious limitation on the existing Grid infrastructure, but currently we are developing the solution that enables transparent proxy connections to the workers via the front cluster node. (2) web and grid services. Below we will concentrate on the architecture of the workflow engine starting from modules as workflow components and finishing with the description of the components the engine is built of.

The VLAM-G environment is constructed from two core components: the graphical user interface (VL-GUI) and the Run-Time System (RTS) – the engine which prepares and executes the workflow, handles the intermediate data and allows the monitoring and the online control. The VL-GUI is used to create a complex scientific experiment interactively by composing a workflow, setting the parameters for each workflow component before and at runtime, and transferring the workflow description to the engine using standard SOAP protocol.

In the following sub-sections, we describe the architecture of the RTS which consists of the RTS Manager (RTSM), the RTSM factory, and the Resource Manager.

3.2.1. The workflow modules

Modules are the core composition elements of the VLAM-G framework. A module is an entity which represents a task to be executed on the Grid. It can be a specially developed application, a web service, or a legacy application. A workflow is created from a set of modules. The modules have input and output ports, which are used to compose the workflow by connecting the output of a module to the input of other modules.

VLAM-G workflows often target coordinated and simultaneous distributed processing of streaming data. Assuming this, a model of a module can be represented as the sequential retrieval of data from input ports, data processing and distribution of the result data to output ports. The execution is performed in a continuous fashion, so that these steps are repeated until interrupted or until the incoming data stream is closed.

A module represents a task being executed on the Grid and from the design point of view it consists of a processing part and a service part. The processing part represents the application itself: the code developed using the VLAM-G module API, a call of a web/grid service, or a launch of a legacy application. The service part provides the interface and the basic facilities for runtime control and the management of data transfers between modules. All available modules are stored in a repository, the creation and deployment of a new module is simple and straightforward and will be explained in the course of the paper.

3.2.2. The RTS architecture

Figure 1 shows a high level overview of the VLAM-G workflow engine: the RTS. The main components of the system are the RTSM Factory, the RTSM and the Resource Manager (RM). The scenario of a workflow enactment using the RTS is the following:

– The RTSM Factory creates an instance of the RTSM, the VLAM-G workflow engine. The RTSM instance is responsible for the given workflow execution. Multiple simultaneous executing workflows are supported. The engine takes as input a directed acyclic graph representing the dataflow of the workflow where the nodes correspond to the modules (workflow components) and the edges reflect the data streams.
– The computing resources where the modules are scheduled can be specified explicitly or retrieved from the RM. If the execution host of a module is not specified then the RM searches for optimal resources for executing the workflow by utilizing the information on module resource requirements. The RM handles the discovery, location, and selection of the necessary resources according to the VLAM-G module requirements. It maps the application tasks to the appropriate resources to optimize the workflow performance, utilizing a number of algorithms and scheduling techniques [4].
– After the resources have been selected, the workflow becomes fully concrete, and the RTSM schedules the workflow components using the Globus job submission mechanism (with non-web service modules), connects the module ports according to the workflow description and sets the module parameters to their default values.
– The RTSM starts the workflow and monitors the execution of each of the VLAM-G modules in this workflow.

3.2.3. VLAM-G resource management

To create an optimal schedule for a meta-application, a data-driven workflow in this case, we need to assign all the components composing the workflow to appropriate resources with the goal of minimizing the cost of the workflow execution. For this, each module in the VLAM-G framework is provided with a module description file that contains information about module resource requirements, including a default execution location which may be left blank.

Before creating an RTSM instance, the RTSM Factory contacts the RM. In turn, the RM processes the description of the workflow, with execution location missing for some modules, together with the requirements of the modules. The RM performs scheduling decisions based on the application information, the available resources information, and cost and application models (Fig. 2). The application information includes the requirements that define the quality of the

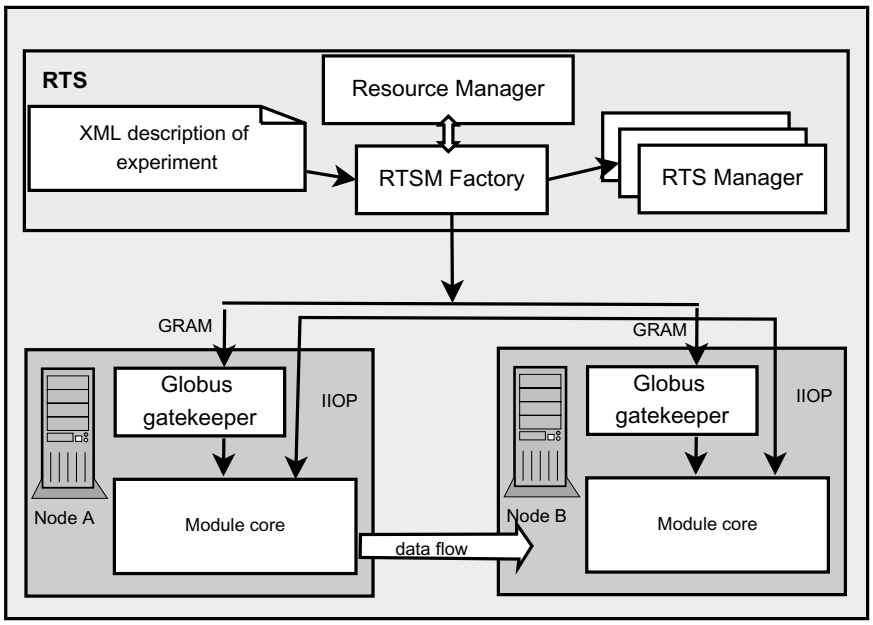

Fig. 1. Run-Time System Architecture.

service requested by the modules. These requirements include the amount of memory needed, the approximate number of processing cycles (i.e. processor load), the storage and the communication load between modules. VLAM-G uses a RSL-like language to specify these requirements (RSL is a resource specification language used in the Globus toolkit to specify the job to be submitted to the Grid Resource Allocation Manager [3]). The resource information is obtained from the Grid Monitoring and Discovery Service (MDS) [3], which also provides forecasts of the resource state from the Network Weather Service (NWS) [2]. This helps to estimate the resource load in the specified time frame in the future and to model the application performance. The cost and application models are used by the resource manager to evaluate a set of candidate schedules for the application. The cost model refers to the execution time of the workflow as the performance metric, and the application model specifies how the execution of the workflow is simulated. The application model is usually represented as a DAG which can be executed in a pipeline or concurrently, and the execution performance depends on the application and on the resource information.

The RM uses several types of heuristics and simulated annealing to achieve sub-optimal schedules based on a performance metric (generally the overall execution time is evaluated). For simulated annealing the RM tries a number of candidate schedules and simulates the execution of the workflow with given requirements on given resources. The results are estimated using a cost model, resource state information, and predictions. The algorithm converges to a sub-optimal schedule which is transferred to the RTSM Factory. Different types of meta-scheduling algorithms (heuristic algorithms and simulated annealing), the evaluation results, and analysis are discussed in detail in [4].

3.2.4. VLAM-G interactive module control

Each VLAM-G module may have parameters that can be monitored and changed during runtime by the module itself or by the user. A parameter in the VLAM-G context is a named entity with a value that can be changed either from within the module code or from outside, i.e. from the VL-GUI. A VLAM-G parameter can be compared to an environment variable of an operating system. This feature allows the users to interact directly with every module composing the application workflow, thus the execution of modules can be controlled on the fly without the need to stop and restart the whole workflow. A parameter is usually associated with a metric that needs to be controlled or is used to monitor the internal state of a module during execution.

3.2.5. Legacy application support

We define a legacy application (LA) as follows: An LA is an application that has no source code available and cannot be modified. LAs are modeled as black

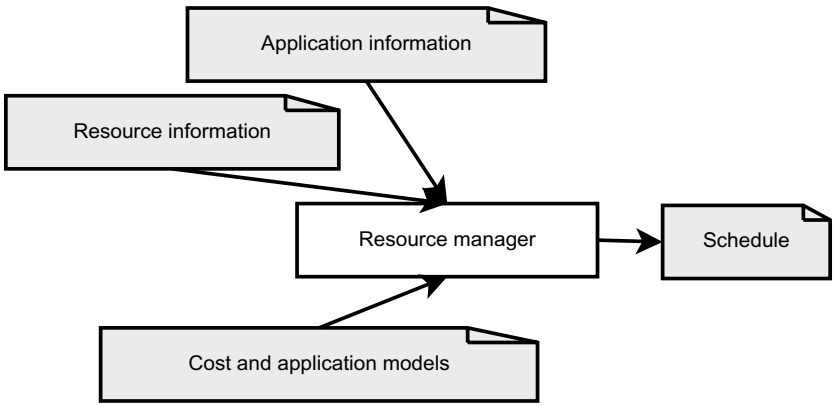

Fig. 2. VLAM-G Resource Management.

box applications exchanging data files; both the input and output data needed for the execution of a LA is assumed to be a set of files. This model can be mapped directly to the VLAM-G module concept because input and output ports can be bound to the exchanged files. For the legacy applications, a generic wrapper has been developed (Fig. 3).

Each LA has a configuration file where all input and output data files are described and bound to the input/output ports. The LA wrapper allows the RTSM to instantiate the LA in the same way as it instantiates a native VLAM-G module. The LA wrapper reads the data from input ports and stores them in the "sandbox" (a specially created directory for a module to keep all the files related to its execution) as files, then executes the LA code. The LA produces a set of output files stored again in the sandbox after processing the data. After the LA finishes the execution, the output files are sent to the associated output ports. This process is repeated until the stream of input files is over. Thus the processing is performed in a loop fashion: continuously repeating the retrieval of input files, the processing by the LA and the transfer of the output files, thus emulating the streams of input and output data.

The same technique is also applied to RPC style web services. Web services expose their operations in WSDL description. Each operation defined in this WSDL can be imported as a VLAM-G module by applying a similar approach as the one used by an LA. Thus a web service with several operations can be imported as a collection of VLAM-G modules. In turn, each of the parameters of every operation described in the web service WSDL is imported as an input port of the corresponding module, and its return values are described as output ports. VLAM-G provides a generic wrapper for this kind of web service operations The web service wrapper collects data from input ports and stores it in input files. A generic client for web services performs the web service calls, setting up operation parameters based on input files, and stores the result in an output file. Finally this output file is sent to the next module via the output port.

Other projects also address the problem of using legacy code in Grid environment. GEMLCA [35] enables the deployment of legacy code applications as Grid services without the need of code re-engineering. Similarly to VLAM-G, the legacy code here is also provided as a black box with specified input and output parameters and environment requirements. The difference is that VLAM-G wraps an LA not as a web-service but as an entity supporting implicit data streaming.

4. VLport library: Design and implementation

The VLport library (also referred as `libvlport` library) is a part of VLAM-G environment and is developed at the University of Amsterdam to support execution of data driven workflows. The `libvlport` library is responsible for maintaining the stream of data from an output port of a module to an input port of another module running on a different computing node. The library is a key component in moving intermediate data produced at a given step of the application workflow to the next step, regardless of the physical location of two processes.

The `libvlport` provides the runtime environment for any VLAM-G module. It offers a basic API for the creation of I/O ports, changing the parameters of the VLAM-G modules, and other utility functions. From the implementation point of view the library provides a class, which must be used as base for developing

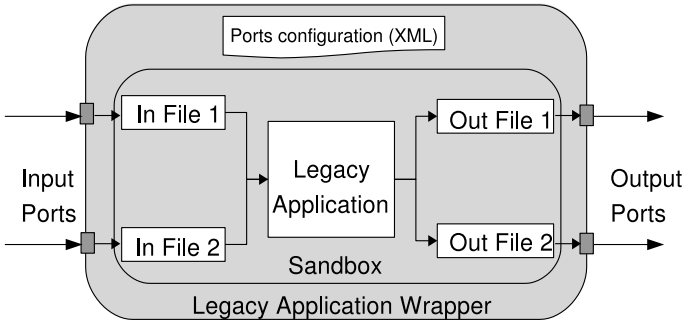

Fig. 3. Legacy application wrapper.

any VLAM-G module. A module developer implements the abstract method `vlmain` (Fig. 4 and listing 1), which contains all computational logic of the module. The RTSM instantiates, connects, and executes all modules composing the workflow. The input and output ports have a simple interface compatible with standard C++ streams. Legacy applications and web services are wrapped and represented to the workflow system as VLAM-G modules as well. Interfaces to Java and Python languages are provided so that native VLAM-G modules can be developed using these languages as well.

Listing 1: Example of a native VLAM-G module

```
#include <vlapp.h>
#include <fstream>

class MyVLApplication : public VL::
VLApplication
{
public:
    MyVLApplication(int argc, char
    **argv)
        : VL::VLApplication(argc,
        argv),
        log("test_vlapp.log")
    {
        myOstream = createDefaultO
        Port("port_out");
    };
    virtual ~MyVLApplication()
    {
        delete myOstream;
    };

    virtual std::ostream& getLog()
    {
        return log;
    };
    int vlmain(int argc , char
    **argv)
    {
        std::ifstream fstr("File.
        orig.dat");
        time_t t1;
        time(&t1);
        *myOstream << fstr.rdbuf();
        getLog()  << "Time:" << time
        (NULL)-t1 << std::flush;
return 0;
    };
private:
    VL::vostream  *myOstream;
    std::ofstream log;
};


int main(int argc , char **argv)
{
    globus_module_activate(GLOBUS_
    COMMON_MODULE);
    globus_module_activate(GLOBUS_
    IO_MODULE);
    try
    {
        MyVLApplication app(argc,
        argv);
        app.run();
    }
    catch(VL::Exception *e)
    {
        std::cerr << e->what();
        delete e;
    }
    globus_module_deactivate(GLOBUS_
    IO_MODULE);
    globus_module_deactivate(GLOBUS_
    COMMON_MODULE);
    return 0;
}
```

The library provides a number of utility functions that gives VLAM-G module developers easy access to all GASS data sources using GridFTP, FTP, HTTP and HTTPS protocol. The data streamed to/from ports can be serialized to the eXternal Data Representation (XDR) [42]. This makes it possible to exploit heterogeneous computational resources. However, module developers can also work with a raw stream, which is similar to the network sockets. The

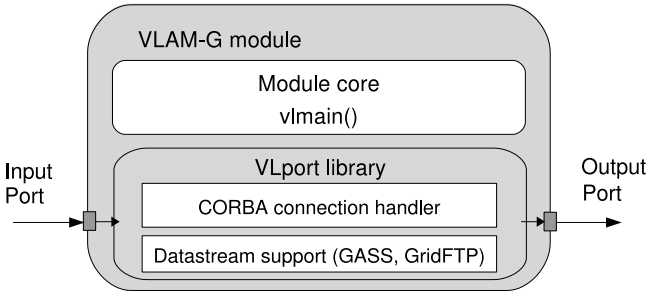

Fig. 4. Port library component architecture.

RTSM establishes all connections, a developer works with opened streams like any C++-compatible streams (i.e. iostream). The communication is performed using Globus IO library [7] and Globus security mechanisms including authentication and authorization of connections, and possibly encryption of the data transferred.

The RTS performs internal module control by using CORBA technology. CORBA has been selected as a stable and mature technology, not as resource- demanding as web services, to enable the management of centrally controlled distributed objects. CORBA is considered as a standard technology for intranet communications, while web services are mostly used in internet scale. RTS manages a set of modules on the intranet level, while clients like other components of VLAM can access RTS via web service interface. Typically one RTS controls the execution of a single subworkflow within a cluster, and composition of several RTS's building together a complex multi-cluster workflow is supported.

For the RTS, each module is represented as a CORBA object [5]. Each instance of a module has IIOP CORBA reference and can be accessed remotely. In order to avoid firewall problems, the library chooses TCP ports from a range specified in GLOBUS_TCP_RANGE variable. Thus, a module can be instantiated even behind a firewall.

The life cycle of a module can be described as follows:

1. *Initializing a module:* The RTSM instantiates a module on a grid-enabled (using the GRAM protocol) or local node. All necessary environment variables are set for proper module initialization.
2. *Creating input/output ports:* During the initialization stage the input and output ports are created. The number of ports is predefined for a module and can not be changed during the runtime.
3. *Registering a module:* The module contacts the RTSM and registers itself. The RTSM keeps sending keep-alive messages to the module during the runtime. If acknowledgment is not received during a predefined timeout the module is considered crashed.
4. *Connecting modules:* The RTSM connects modules with each other according to the dataflow; an output port can be connected to many input ports (one-to-many relationship implemented as a set of standalone connections);
5. *Scheduling a module:* The RTSM schedules the modules for execution (the method `vlmain` is executed);
6. *Exchanging data with other modules:* The module reads the data from input ports, processes it and writes results to output ports: loop fashion execution;
7. *Module termination:* When a module exits (i.e. input port has been closed by the previous module) it returns from `vlmain` method. All pending buffers are flushed, and ports are closed. The module unregisters itself from RTSM registry and exits.

The control of the parameters of a VLAM-G module is implemented as CORBA remote procedure calls. All the modules in the worklfow are controlled by the RTSM using IIOP protocol.

To utilize various programming languages such as Java and Python a set of wrappers has been developed. The wrappers translate the calls from the target language to the C++ libvlport library. Figure 6 shows the Java wrapper library based on Java Native Interface (JNI). The VLAM-G module core and the wrapper are executed in the same Java virtual machine, the wrapper functions translate the calls to the libvlport library. Thus a Java module has the same functionality and life-cycle as a native module. The Python wrapper employs similar techniques using the SWIG (simplified wrapper and interface generator) to generate a wrapper around the libvlport library. Similar solutions for

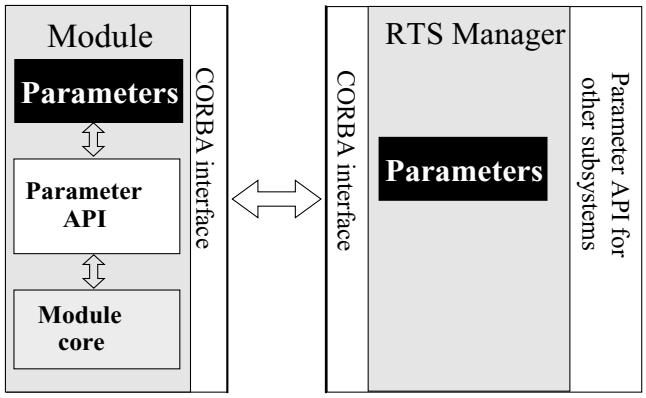

Fig. 5. Module parameter interface.

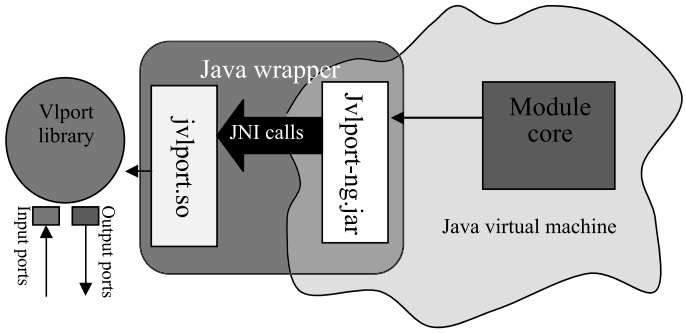

Fig. 6. Java wrapper.

code wrapping have been employed in other projects, for example Jopera [29] provides a wrapper for Java snippets, small blocks of Java code usually needed to preform small computation. It also provides a wrapper for Unix applications (legacy applications) using the shell command line and the pipe-based inter-processes communication.

The VLAM-G port library is developed in C++ and is implemented as a dynamic library for UNIX systems. It uses the threaded versions of Globus libraries and OmniORB – a free high performance ORB for C++ [5]. The RTSM is a part of the VLAM-G front-end and is developed in Java. It uses Commodity Grid Kit for Java [6] and Community OpenORB Project [8].

5. Performance evaluation

The RTS library is designed to provide maximal performance for distributed applications while hiding all low level details from the application developers. It must thus provide a throughput comparable with the standard protocols. In this section we compare the performance of the library with the performance of standard data transfer tools included to the Globus toolkit, and measure the introduced overhead.

We evaluated the dependency of the data transfer performance on the data block size using both standard Globus utilities and VLport library. The data transfers took place between the nodes of clusters connected through a high-speed WAN (wide area network). The clusters are part of the ASCI Supercomputer 2 (DAS 2), which is a multi-cluster system distributed over different universities in the Netherlands. Each cluster node contains: two 1-Ghz Pentium-IIIs, at least 1 GB RAM, a Myrinet interface card, a Fast Ethernet interface (on-board). The nodes within a local cluster are connected by a Myrinet-2000 network, which is used as high-speed interconnect, mapped into user-space. In addition, Fast Ethernet is used as OS network (file transport). The five local clusters are connected by the Dutch university Internet backbone [11]. To measure the overhead introduced by the VLAM-G library, the average throughput of the link was evaluated with the help of standard Globus tool ‘globus-url-copy’ using GridFTP protocol. Figure 7 shows the data transfer

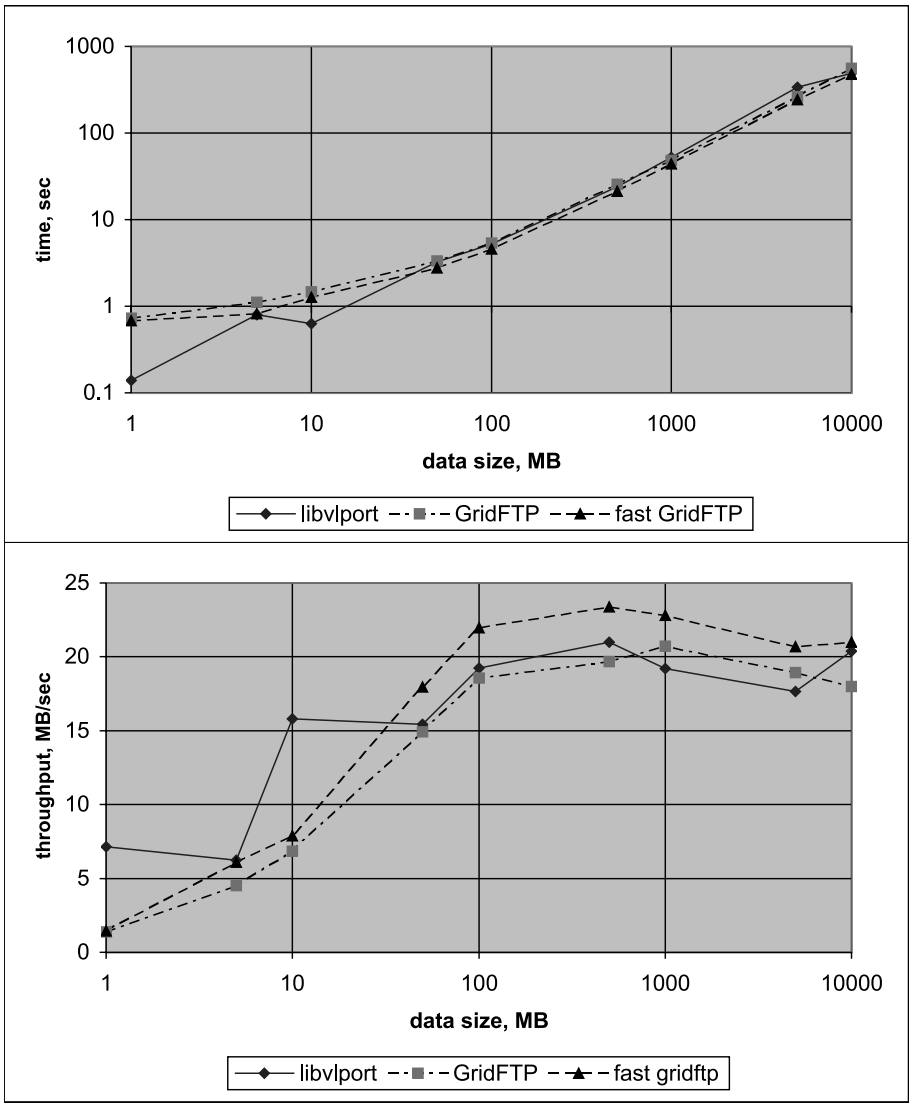

Fig. 7. Average performance of the RTS library on WAN compared with standard Globus data transfer tool.

rate as a function of the data block size (average of 10 measurements per each data-block with the deviation not exceeding 5 percent). Compared to the throughput obtained with the VLAM-G RTS library, the globus-url-copy shows slightly better performance for the test with parallel stripes (fast GridFTP) and almost the same outcome in the case of a standard GridFTP protocol. The maximum data throughput for large data blocks is about 20 MB/sec for the library. Since the RTS Library is designed to support data streams, the performed tests show encouraging results as the speed of the data transfer achieved using the library is comparable to those achieved using standard Globus tools.

6. A bioinformatics use case

Here we present the SigWin-detector workflow, a concrete example of how VLAM-G can create an environment where biologists can perform their (computational) experiments on the grid without having to deal with the complexities of the underling grid middleware, so they can focus on the biological aspects of their experiments. SigWin-detector was presented at the 2nd IEEE international conference on e-Science and Grid computing [41].

SigWin-detector is a generic workflow that can analyze any ordered sequence of values, spanning from gene expression data to local time series of tempera-

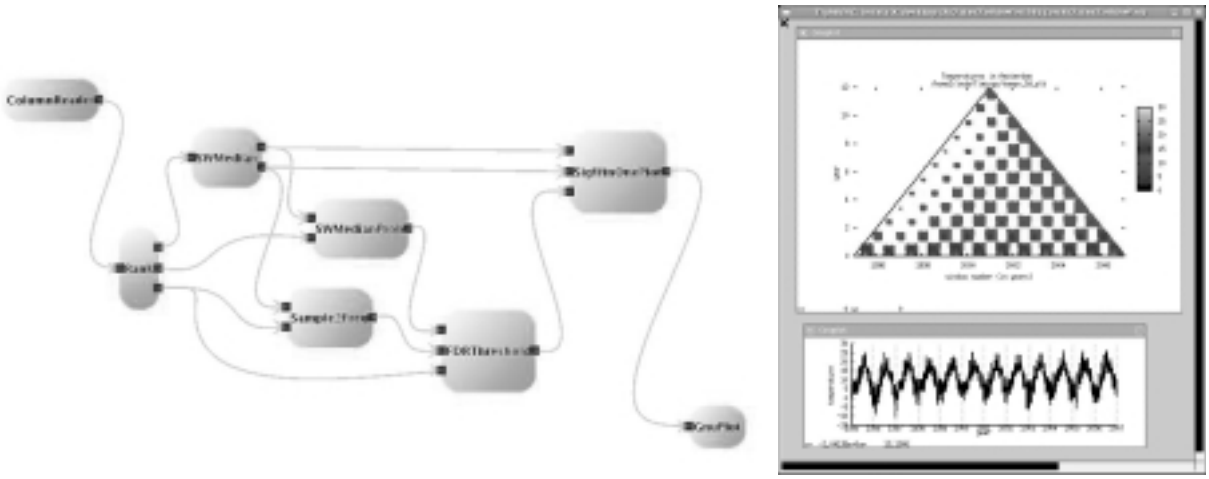

Fig. 8. Left: SigWin detector basic workflow using the VLAM-G workflow composer. Right: graphical output of the basic workflow for the time series of temperaures in Amsterdam from 1995 up to 2006.

ture. It is used to identify regions in the sequence where the median value is higher than could be expected by chance if the ordering of the values was not relevant. For example, in a gene expression sequence it identifies regions of highly expressed genes, while in time series of temperature it identifies hot periods such as summers or heat waves. The workflow consists of modules that implement a generalized and improved version of the RIDGE method [36], a method originally designed to identify Regions of Increased Density of Gene Expression (RIDGE(s)) in profiles of the gene expression activity of a cell plotted along the chromosomes, so-called transcriptome maps. SigWin-detector workflow can be operated interactively in a grid environment. It takes as input an ordered sequence (a transcriptome map), computes sliding window medians, and identifies as significant windows (RIDGEs) the sliding windows with a median value above a certain *false discovery rate* (*FDR*) threshold, c.f. [38]. Those windows are called *significant windows*, or RIDGES in the special case that the input sequence is a transcriptome map. The SigWin-detector basic workflow (Fig. 8, left) consists of the following VLAM modules: (1) ColumnReader, reads the input sequence; (2) Rank, ranks it; (3) SWMedian, computes sliding window medians for a given range of odd window sizes; (4) Sample2Freq, generates a frequency count from the sliding window median data; (5) SWMedianProb, generates a theoretical probability density for the sliding window median data from the ranked sequence; (6) FDRThreshold, applies a false discovery rate (FDR) procedure that compares the obtained frequency count with the theoretical probability to obtain FDR thresholds for each window size; (7) SigWindow, selects windows above the FDR threshold; (8) GnuPlot, displays the results graphically.

Each module performs specific tasks that can be fine-tuned by using the module's parameters. For example, the parameters of module SWMedian are used to define the range of window sizes to be computed.

The workflow uses the automatic streaming feature of VLAM-G to assemble a pipeline that streams the data generated for each window size to the next module as soon as it is available: Suppose we want to detect significant windows for window sizes s_1 up to s_4. While the SWMedian module is generating sliding window medians for window size s_4; modules Sample2Freq and SWMedianProb will be producing the data for window size s_3; module FDRThreshold will be computing the thresholds for window size s_2; and module SigWin will be selecting the significant windows for window size s_1. Streamig the data also saves the work of writing intermediary data to files. Alternatively, if this data is important, we can add a module that will write it to a file.

Figure 8, right shows the graphical output of the basic workflow for a time series of temperaures in Amsterdam from 1995 up to 2006. The triangular graph displays the significant windows for all window sizes showing the obvious cyclic pattern of the seasons. Each row represents a window size s, where s is an odd number ranging from 1 day to $N = 11$ years. Each column represents a sliding window number (ranging from $(s-1)/2$ to $N-(s-1)/2$), hence the triangular form. Each significant window is identified by a point

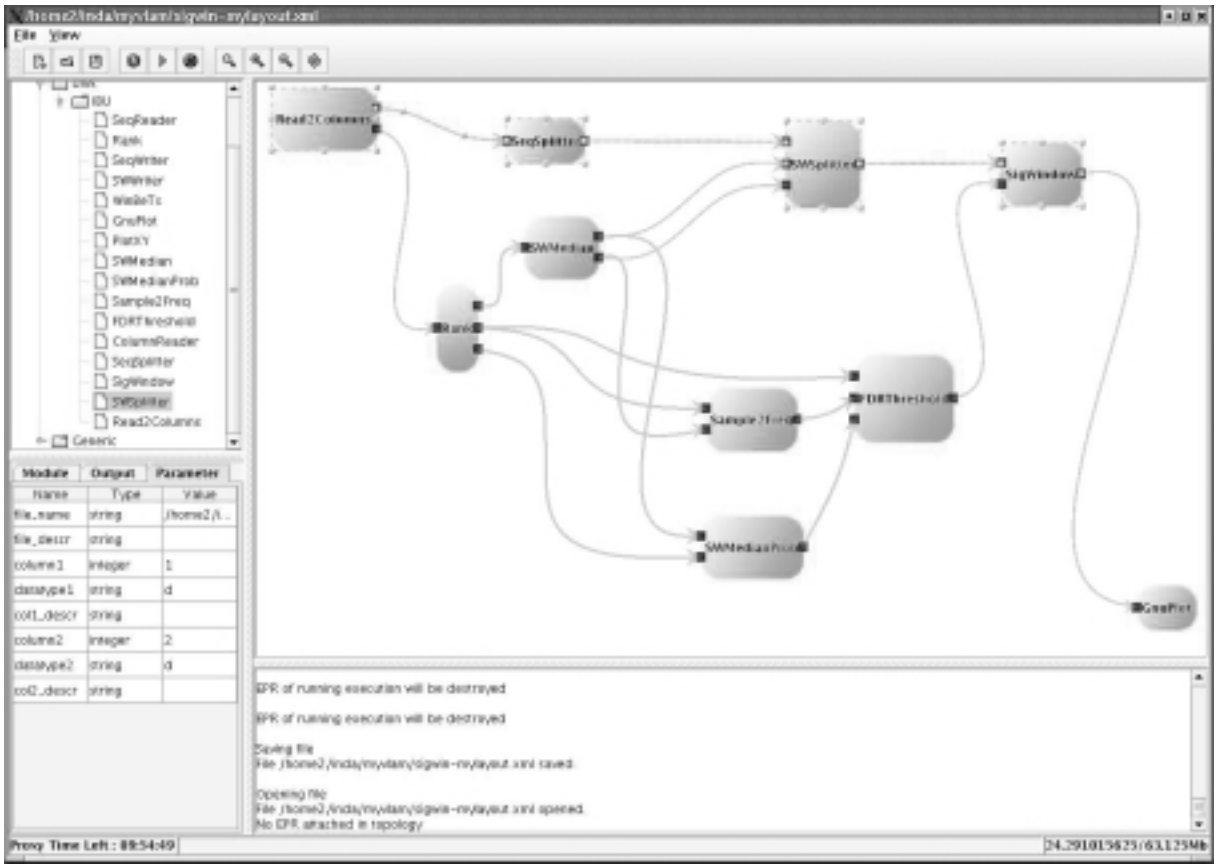

Fig. 9. VLAM-G environment and SigWin detector workflow adapted for displaying significant windows per subsequence. The new modules are: (A) Read2Columns, (B) SeqSplitter, (C) SWSplitter, and (D) SubSigWindow.

in the graph, and the colors represent the actual sliding window median expression value.

The basic workflow can be altered by substituting, deleting, or adding modules. For example, if we do not want graphical display, the graphics module can be dropped (this is useful for batch processing). Another useful alteration is to modify the workflow so that it identifies significant windows in subsequences of the original sequence. This subsequence could be a year or a month in the case of time series, or a chromosome in the case of transcriptome maps. In the later case, discovering RIDGEs per chromosome is more sensible, since the ordering of the chromosomes is arbitrary; furthermore, this also reduces the computational work, since the maximum number of genes in a chromosome is considerably smaller than the total number of genes. Figures 9 and 10 show the modified workflow and its graphical output for the case of human trancriptome map complied in [36].

Besides the graphical output supported by an interactive virtual display, the VLAM-G environment provides access to standard output and standard error streams of running modules at runtime. SigWin modules can generate results with various levels of detail. The logging detail level is a module parameter that may be controlled at runtime. Thus, depending on his needs, the user can control the amount of generated and printed data at any moment with a simple change in logging parameter values of modules executing remotely.

Implementing SigWin-detector using VLAM-G allowed us to take advantage of some of the features of the VLAM-G workflow namely: (a) the interactive creation and execution of workflows in a grid environment; (b) interactive control of module execution using parameters (c) the automatic redirection of the remote graphical output to the end-user default screen; (d) the possibility to adapt the workflow to meet the user's specific needs; (e) the ability to run an experiment in batch mode by calling the runtime system from a script directly.

Furthermore, VLAM-G's modular design discloses the structure of an experiment to biologists in a

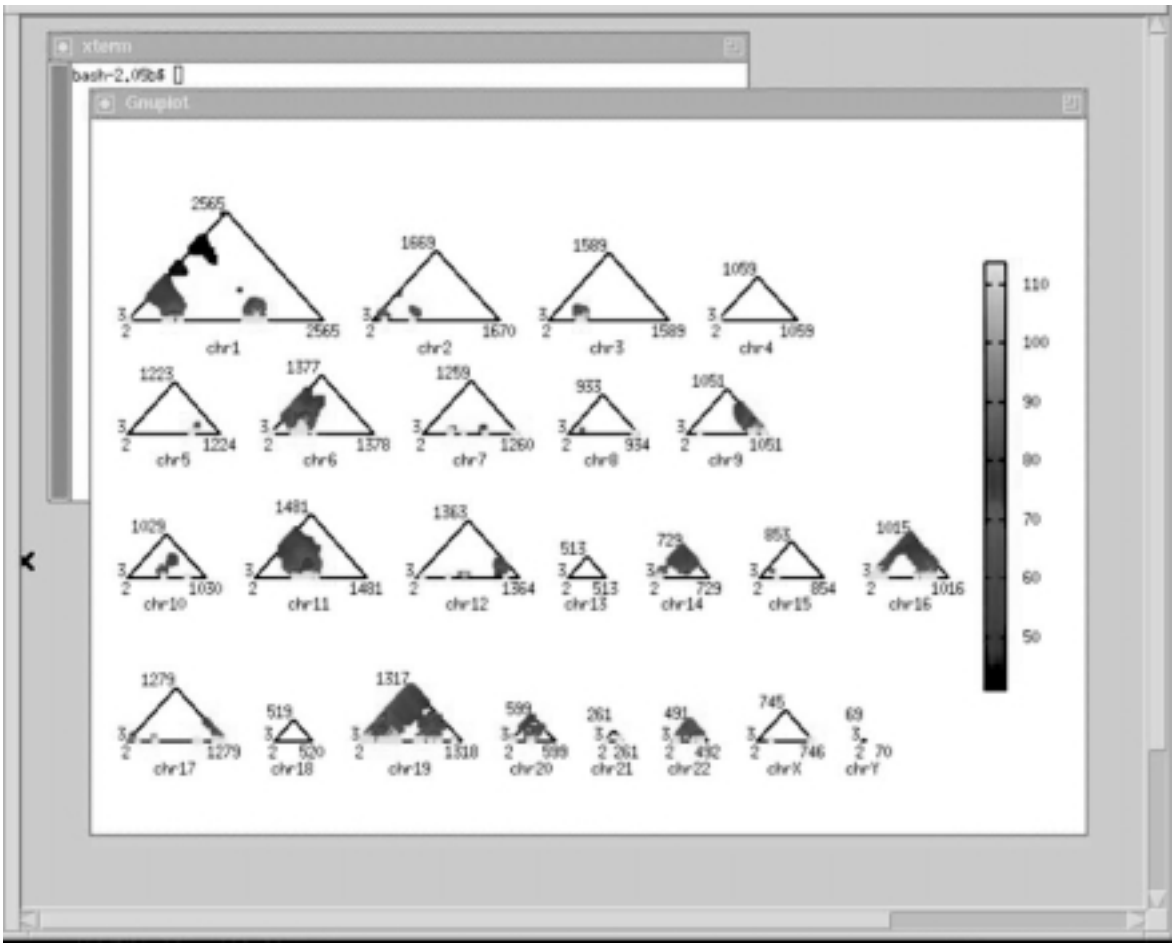

Fig. 10. Graphical output of the modified SigWin-detector workflow taking as input the human transcriptome map compiled in [36].

more testable way than through written publications and poorly reproducible scripts. This facilitates optimization of parts of the workflow and enables experimentation with variations of the original design. Also, the workflow itself can be used as a 'module' in more elaborate experiments.

7. Conclusions and future work

In this paper we present VLAM-G – a data-driven workflow management system, and the Run-Time System, its engine. This engine allows the execution of data-driven workflows in a transparent way on the available Grid resources. A VLAM-G workflow is composed of entities called modules that interface native VLAM-G applications, web services and legacy applications even if the source code is not available for modification. Interactive control of the execution is provided: the end-users can interact with any workflow component at runtime via a simple parameter interface or by accessing a virtual display with remote graphical output. The API for native VLAM-G modules is provided by VLport runtime libraries which have been designed to support different languages: C/C++, Java and Python. A generic wrapper provides the means to port an existing application or a web service as a standard VLAM-G workflow component.

The VLAM-G workflow management system can be used for distributed applications requiring direct data streaming between the remote components, e.g. semi-realtime applications, remote devices access and control etc. The performance evaluation showed that the overhead brought by intermediate VLport library is negligible. The analysis of the presented use case shows the advantages of the VLAM-G environment in the development of distributed data-driven applications.

In the current version, the parameters of VLAM-G modules are not shared, which restricts the interaction of the modules to simple input/output ports. In further versions we plan to introduce a rule-based system that will monitor the status of the workflow at runtime and change parameters of a given module depending on the workflow state. Currently an external centralized information system is used to keep track of the VLAM-G module description and characteristics. We plan to incorporate a discovery system based on peer-to-peer network that will provide up-to-date information about available modules and resources.

We are also investigating how additional background knowledge about the data can be automatically incorporated in the system [37]. This information will be needed for the interpretation of increasingly complex experiments made possible by Virtual Laboratory.

Acknowledgments

We would like to thank Dr. Marcel van Batenburg for fruitful discussions and the bioinformatics group of Dr. Antoine van Kampen for making the human transcriptome map data available to us.

This work was carried out in the context of the Virtual Laboratory for e-Science project (www.vl-e.nl). Part of this project is supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science (OC & W) and is part of the ICT innovation program of the Ministry of Economic Affairs (EZ).

This work was part of the BioRange program of the Netherlands Bioinformatics Centre (NBIC), which is supported by a BSIK grant through the Netherlands Genomics Initiative (NGI).

References

[1] H. Afsarmanesh, R.G. Belleman, A.S.Z. Belloum, A. Benabdelkader, J.F.J. van den Brand , G.B. Eijkel, A. Frenkel, C. Garita, D.L. Groep, R.M.A. Heeren, Z.W. Hendrikse, L.O. Hertzberger, J.A. Kaandorp, E.C. Kaletas, V. Korkhov, C.T.A.M. de Laat, P.M.A. Sloot, D. Vasunin, A. Visser and H.H. Yakali VLAM-G: A Grid-Based Virtual Laboratory, *Scientific Programming* **10**(2) (2002), 173–181.

[2] R. Wolski, N. Spring and J. Hayes, The Network Weather Service: Distributed Resource Performance Forecasting Service for Metacomputing, *Journal of Future Generation Computing Systems* **15**(5–6) (October 1999), 757–768.

[3] K. Czajkowski, S. Fitzgerald, I. Foster and C. Kesselman, Grid Information Services for Distributed Resource Sharing, *The Tenth IEEE International Symposium on High-Performance Distributed Computing* (*HPDC-10*), IEEE Press, August, 2001.

[4] V. Korkhov, A. Belloum and L.O. Hertzberger, Evaluating Meta-scheduling Algorithms in VLAM-G Environment, *Tenth Annual Conference of the Advanced School for Computing and Imaging* (*ASCI*) (June, 2004).

[5] omniORB, Free High Performance ORB. http://omniorb.sourceforge.net.

[6] Commodity Grid Kits, http://www.globus.org/cog/.

[7] The Globus Alliance, http://www.globus.org/.

[8] The Community OpenORB Project, http://openorb.sourceforge.net.

[9] Virtual Laboratory for E-Science, http://www.vl-e.com.

[10] Netperf: A Network Performance Benchmark. Information Network Division, Hewlett-Packard Company, 1995 http://www.netperf.org/netperf/NetperfPage.html.

[11] The Distributed ASCI Supercomputer 2 (DAS-2) http://www.cs.vu.nl/das2/.

[12] S. Krishnan, K.K. Baldridge, J.P. Greenberg, B. Stearn and K. Bhatia, An End-to-end Web Services-based Infrastructure for Biomedical Application. Grid 2005, 6th IEEE/ACM International Workshop on Grid Computing, Nov 2005.

[13] G. Kickinger, J. Hofer, A.M. Tjoa and P. Brezany, Workflow Management in GridMiner. In Proceedings of the 3rd Cracow Grid Workshop, Cracow, Poland, October 2003.

[14] T.M. Nguyen, A.M. Tjoa, G. Kickinger and P. Brezany, Towards Service Collaboration Model in Grid-based Zero Latency Data Stream Warehouse (GZLDSWH) Proceedings of the 2004 IEEE International Conference on Services Computing (SCC 04).

[15] D. Gannon, S. Krishnan, A. Slominski, G. Kandaswamy and L. Fang, *Building Applications from a Web Service based Component Architecture*, Proc. of the Workshop on Component Models and Systems for Grid Applications, June 26, 2004 held in Saint Malo, France. Springer, 2005, to appear. ISBN: 0-387-23351-2.

[16] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher and S. Mock, *Kepler: An extensible system for design and execution of scientific workflows*, in proceedings of 16th International Conference on Scientific and Statistical Database Management, 2004, 423–424.

[17] J. Yu and R. Buyya, A taxonomy of scientific workflow systems for grid computing, *SIGMOD Record* **34**(3) (2005).

[18] B. Wilson, B. Tang, G. Manipon, D. Mazzoni, E. Fetzer, A. Eldering, A. Braverman, E.R. Dobinson and T. Yunck, GENESIS SciFlo: scientific knowledge creation on the grid using a semantically-enabled dataflow execution environment, Proceedings of the 17th international conference on Scientific and statistical database management, 2005, Santa Barbara, CA, 83–86.

[19] J. Kommineni, D. Abramson and J. Tan, *Communication over a Secured Heterogeneous Grid with the GriddLeS runtime environment*, 2nd IEEE International Conference on e-Science and Grid Computing. Dec. 4–6, 2006, Amsterdam, Netherlands.

[20] S. Pallickara and G. Fox, *NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids*, Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003. pp. 41–61. Lecture Notes in Computer Science 2672 Springer 2003, ISBN 3-540-40317-5.

[21] K. Benedyczak, A. Nowinski, K. Nowinski and P. Bala, *UniGrids Streaming Framework. Enabling streaming for the new generation grid*, ICM, PARA 2006 Umea Sweden, 18–21 June 2006.

[22] J. Blower, K. Haines and E. Llewellin, *Data streaming, workflow and firewall-friendly Grid Services with Styx*, Proceedings of the UK e-Science All Hands Meeting 19–22 September 2005.

[23] M. Aldinucci, M. Coppola, M. Danelutto, M. Vanneschi and C. Zoccolo, *ASSIST as a Research Framework for High-performance Grid Programming Environments*, In Jose C. Cunha and Omer F. Rana, editors, Grid Computing: Software environments and Tools, chapter 10, Springer, Jan. 2006, 230–256.

[24] J. Delaney, J. Orcutt, M. Abbott, L. Smarr and E. Lazowska, *The NSF Laboratory for Ocean Observatories Knowledge INtegration Grid* (*LOOKING*), American Geophysical Union, Fall Meeting 2005, abstract IN21B-1182

[25] T. Fahringer, A Jugravu, S. Pllana, R. Prodan, C. Seragiotto Jr. and H. Truong, Askalon: a tool set for cluster and Grid computing, *Concurrency and Computation: Practice and Experience* **17** (2005), 143–169.

[26] T. Oinn, M. Greenwood, M. Addis, M.N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M.R. Pocock, M. Senger, R. Stevens, A. Wipat and C. Wroe, *Taverna: Lessons in creating a workflow environment for the life sciences* Concurrency and Computation: Practice and Experience, Volume 18 Issues 10, Grid Workflow Special Issue, August 2005, 1067–1100.

[27] I. Taylor, I. Wang, M. Shields and S. Majithia, Distributed computing with Triana on the Grid, *Concurrency and Computation:Practice and Experience* **17**(1–18) (2005).

[28] G. Allen, K. Davis, K. Dolkas, N. Doulamis, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzyski, J. Pukacki, T. Radke, M. Russell, E. Seidel, J. Shalf and I. Taylor, Enabling Applications on the Grid: A GridLab Overview, *International Journal of High Performance Computing Applications: Special issue on Grid Computing: Infrastructure and Applications* (August 2003).

[29] C. Pautasso and G. Alonso, JOpera: a Toolkit for Efficient Visual Composition of Web Services, *International Journal of Electronic Commerce* (*IJEC*) **9**(2) (Winter 2004/2005).

[30] F.J. Seinstra and J.M. Geusebroek, *Color-Based Object Recognition on a Grid*, Proceedings of the 9th European Conference on Computer Vision (ECCV 2006) Workshop on Computation Intensive Methods for Computer Vision (CIMCV 2006), Graz, Austria, May 7–13, 2006. Copyright ©2006 Springer-Verlag. (PS).

[31] S.M. Charters, N.S. Holliman and M. Munro, *Visualisation on the Grid: A Web Service Approach*, UK e-Science All Hands Meeting 2004 (September 2004).

[32] G. Fox, G. Aydin, H. Bulut, H. Gadgil, S. Pallickara, M. Pierce and W. Wu, *Management of Real-Time Streaming Data Grid Services*, Concurrency and Computation: Practice and Experience, Special Issue from Grid and Cooperative Computing 4th International Conference November 30 to December 3 2005 Beijing China, 2006

[33] P. Kacsuk, G. Dozsa, J. Kovacs, R. Lovas, N. Podhorszki, Z. Balaton and G. Gombas, P-GRADE: A Grid Programming Environment, *Journal of Grid Computing* **1**(2) (2003), 171–197(27).

[34] T. Tannenbaum, D. Wright, K. Miller and M. Livny, *Condor – A Distributed Job Scheduler*, Beowulf Cluster Computing with Linux, The MIT Press, MA, USA, 2002.

[35] T. Delaittre, T. Kiss, A. Goyeneche, G. Terstyanszky, S. Winter and P. Kacsuk, GEMLCA: Running Legacy Code Applications as Grid Services, *Journal of Grid Computing* **3**(1–2) (June 2005), Springer Science + Business Media B.V., Formerly Kluwer Academic Publishers B.V. ISSN: 1570–7873, 75–90.

[36] R. Versteeg, B.D. van Schaik, M.F. van Batenburg, M. Roos, R. Monajemi, H. Caron, H.J. Bussemaker and A.H. van Kampen, The Human Transcriptome Map Reveals Extremes in Gene Density, Intron Length, GC Content, and Repeat Pattern for Domains of Highly and Weakly Expressed Genes, *Genome Research* **13**(9) (2003), 1998–2004.

[37] M.S. Marshall, L. Post, M. Roos and T.M. Breit, *Using Semantic Web Tools to Integrate Experimental Measurement Data on Our Own Terms*, International Workshop on Knowledge Systems in Bioinformatics (KSinBIT'06), 2006

[38] Y. Hochberg and Y. Benjamini, More Powerful Procedures for Multiple Significance Testing, *Stat Med* **9**(7) (1990), 811–818.

[39] Jr. G. Chin, L.R. Leung, K. Schuchardt and D. Gracio, New Paradigms in Problem Solving Environments for Scientific Computing. Proceedings of the international conference of Intelligent User Interface 2002, San Francisco.

[40] Workshop on workflow management in scientific and engineering applications report, SIGMOD Rec. vol. 26 no. 4, 1997, ISSN 0163-5808, pp. 49–53, DOI http://doi.acm.org/10.1145/271074.271087, ACM Press.

[41] M.A. Inda, A.S.Z. Belloum, M. Roos, D. Vasunin, C. de Laat, L.O. Hertzberger and T.M. Breit, *Interactive Workflows in a Virtual Laboratory for e-Bioscience: The SigWin-Detector Tool for Gene Expression Analysis*, Second IEEE International Conference on e-Science and Grid Computing (e-Science'06), 2006.

[42] R. Srinivasan, XDR: External Data Representation Standard, RFC 1832, DDN Network Information Center, Aug. 1995.

[43] B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel and S. Tuecke, *Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing*, IEEE Mass Storage Conference, 2001.

[44] W. Li, I. Shindyalov, H. Casanova, L. Ang, F. Berman, P. Arzberger, M. Miller, P. Bourne, R. Byrnes, J. Hayes, A. Birnbaum, V. Reyes, A. Shahab, C. Mosley, D. Pekurovsky and G. Quinn, The Encyclopedia of Life Project: Grid Software and Deployment. New Generation Computing, Volume 22, Issue 2, January 2004, 127–136.

[45] A. Dowsey, M.J. Dunn and G.Z. Yang, ProteomeGRID: Towards a High-Throughput Proteomics Pipeline Through Opportunistic Cluster Image Computing for Two-Dimensional Gel Electrophoresis, *Proteomics* **4**(12) (December 2004), 3800–3812.

[46] A. Konagaya, *Trends in Life Science Grid: From Computing Grid to Knowledge Grid*, International Conference in Bioinformatics – InCoB2006 18–20 December 2006, New Dehli, India.