A Practical Component Framework for Development of Scientific Grid Applications

V. Šipková, R. Forgáč, J. Astaloš, V. D. Tran, and M. Dobručky

Ústav Informatiky, Slovenská Akadémia Vied
845 07 Bratislava, Dúbravská cesta 9

Abstract. Software Component Frameworks are widely used standards in commercial business applications. In the last decade this technology is being explored with great interest as a way to build large-scale scientific applications on parallel computers and Grid systems. Nowadays, professional programmers attempt to build complex applications by composing the elements from large collections of predefined and tested units which are made available from several teams of specialists or open source community. The component technology fits very well with the manifold service-oriented Grids, however, the model must allow for a very dynamic control of composition. This paper describes a new component-based framework providing the higher-level components built on top of modern Grid technologies, that allow the easy and efficient composing and deployment of Parameter Sweep Applications onto a Grid platform.

1 Introduction

The computational science community hold huge existing investments in a broad assortment of physics, chemistry, climate, materials, fusion, combustion, radio astronomy, numerical systems, and visualization software. A scientific breakthrough is feasible by combining the best-in-class technologies from different disciplines into an integrated application. This is in practice not trivial to achieve due to codes are using different programming languages, data models, or different standards.

Software component frameworks are widely used standards in commercial business applications. In the last decade this technology is being explored with great interest as a way to build large-scale scientific applications on parallel computers and Grid systems. Due to the high computational complexity involved professionals attempt to build complex applications by composing the elements from large collections of predefined and tested units which are made available from several teams of specialists or open source community. This effort began in 1995 when a small group of collaborators tried to define the concept of the component architecture for scientific applications running on massively parallel systems. Their activity has grown to the establishment of the consortium [1] dedicated to defining and promoting the component specification standard called Common Component Architecture (CCA) [2]. The CCA model is brought about as a means to allow applications to run as components and easily interoperate.
with other components without the need to worry about the language, operating system, or architecture differences.

At the same time the CCA has been developed and tested on the high-performance computers, the first applications have run on multiple resources in Grid environments [3, 4]. The current architectural model of Grid systems is based on the emerging Web services framework OGSA [5, 6], spanning different projects, organizations and application domains. Grid-aware applications make use of coupled heterogeneous resources, such as computers, databases, mathematical libraries and scientific instruments, that are not available at a single site. It is well known that designing and implementing efficient distributed/parallel programs for high-performance computers is still a challenging time-consuming task. Grid computing makes the situation even worse. Grid-aware applications are characterized by several additional dimensions of complexity - they are more heterogeneous, dynamic, and distributed in both space and time.

The CCA paradigm fits very well with the manifold service-oriented Grids [7, 8], however, the model must allow for a very dynamic control of composition, supporting multiple protocols and interoperability mechanisms. In order to deal with the scale and heterogeneity, the component model must be hierarchical, and to enhance programmability, it should allow the use of the structured composition to design new components. To deal with dynamics, the model must be capable of handling reconfiguration. Moreover, the component model should be able to handle components at the infrastructure level in order to benefit from predefined, efficient and reusable service components. Decomposing a scientific application involves splitting the program package, both the code and data, into chunks that can be distributed to Grid resources for proper handling. The realization of this may be accomplished in miscellaneous ways and on multiple structural layers, ranging from the explicit communication approaches such as message passing MPI [9] and Grid remote procedure call GridRPC [10], to Grid workflows [11], bags-of-tasks, and various complex meta-components built on top of the underlying Grid middleware. However, currently, there is little consensus on suitable programming paradigms for Grid application development.

The rest of the paper is organized as follows. Section 2 outlines the motivation of our work. Section 3 gives an overview of similar research efforts and related projects. Section 4 describes the design of our framework, prototype of which is currently being under the development and implementation. The framework has been tested with the real-world application from the area of image processing, this is presented in Section 5. Section 6 concludes the paper with future plans.

2 Motivation

The current Grid middleware infrastructures deliver most of the required functionality for Grid application development, however, they are too complex and require non-trivial knowledge to be used correctly. A promising approach has been proved to supply the user with so-called Application-level Tools [12] that are able to isolate him from the middleware infrastructure and take on all of the
deployment complexity. Our work is targeted to this effort.

We intend to provide a component-based framework built on top of the underlying Grid middleware that allows programmers the easy and efficient porting and development of scientific applications on a Grid platform. We concentrate primarily on an important class of applications called Parameter Sweep Applications (PSAs) which arise in many scientific and engineering contexts. The Grid computing developments have created opportunities for running PSAs on remote networked high-performance resources, but at the expense of greater complexity, as they have become multi-tiered. PSAs need to be executed over large parameter spaces and are typically structured as a bulk of experiments, where in each of which the same program is carried out with a different set of input data. Due to the inherent parallelism this can be done in a distributed fashion thus significantly reducing execution time. Although PS experiments are independent (i.e. do not communicate), many of them can share large input files and typically produce a large suite of output files which must be post-processed and/or moved to a long-term storage. To achieve performance on a Grid platform the input data have to be distributed appropriately and the algorithm for scheduling of independent tasks should be chosen regarding some prediction information about the computation and file transfer times.

3 Related Work

Because of the wide range of scientific component research that addresses CCA and Grids, a variety of developing frameworks have been proposed and realized, both general purpose and domain-specific, with distinct capabilities and restrictions. To list all of them would go beyond the scope of this document. A number of Grid computing environments are described in the paper [13]. The impact that the CCA technology has made on the computational science is highlighted in the survey paper [14]. The SciDAC’s Center for Component Technology for Terascale Simulation Science [15] which has supported the CCA Forum since 2001, favours three production frameworks: Ccaffeine [16], XCAT [17] and SCIRun2 [18]. GrADS [19] (Grid Application Development Software) is perhaps the most comprehensive Grid research project. It investigates an integrated approach to Grid applications development and execution that includes automatic Grid-enabled libraries, compilation and optimization techniques targeting the Grid, runtime scheduling systems, dynamic application monitoring and control, and market-based Grid resource allocation strategies. GridWay [20] is an open-source community Globus project [21, 22]. GridWay Meta-scheduler performs the job execution management and resource brokering on a Grid consisting of distinct computing platforms managed by Globus services.

There are three well-known Grid programming environments related to the developing of PSAs: Nimrod/Nimrod-G [23], NASA ILAB [24], and APST [25, 26]. Nimrod/Nimrod-G (Tools for Distributed Parametric Modelling) is a specialized modelling system designed to explore the behavior of complex parametric experiments. Nimrod-G is a Grid aware version of Nimrod. ILAB (Information
Power Grid Virtual Laboratory) is a software package that automates the process of creating, submitting, and monitoring parameter studies on NASA’s Information Power Grid resources. APST (AppLeS Parameter Sweep Template) is a Grid application environment targeted to PSAs consisting of fixed number of independent tasks. APST-DV is an extension of APST handling divisible load applications. There is some similarity between the approach we plan to apply and the component framework based on the service-oriented middleware VGE (Vienna Grid Environment) [27, 28]. It respects the CCA and models application Web services as distributed components. VGE is utilized in the project GEMSS, which focuses on advanced medical simulation services.

The most of the Grid programming environments are complex systems involving typically much installed software. Several of them are far from the concept of an invisible Grid. In contrast to these, the main goal of our work is to provide a practical, flexible and easily manageable development tool enabling the user to adapt and execute PSAs in a Grid environment without the actually making explicit Grid actions.

4 The Design Overview

In this section we outline the architecture of the framework we have designed. A simplified block diagram of the main parts and their relationships is shown in Fig. 1.

![Fig. 1. Framework Architecture](image)
The framework consists of three major components: the *Data Distributor*, *Task Executor*, and *Output Retriever*. The input of the framework includes an *Application Package* and *Application Descriptor*, and application results are contained in the *Output Data Package*.

**Application Package** consists generally of several files including the application executable file, optional small input files used as command line arguments, and input data files which might be pre-staged on a storage resource.

**Application Descriptor** provides meta-data about the application itself and application demands. The format which the descriptor is of is implementation dependent. For testing purposes we have implemented a preliminary framework prototype using a shell scripting language, therefore the current application descriptor is a simple file of shell type. In the next step the framework is planned to be realized applying the standard Web service technology conforming the CCA model. The framework will be implemented as a Web service constructed on top of an existing Grid middleware. In this case the application meta-data are required to be in the form of an XML document.

**Data Distributor** is responsible for the partition of the input data set into a certain number of subsets assigned afterward to computing elements for separate job execution. We consider PSAs that match the so-called *divisible load model*, i.e. applications those whole input data can be divided into a number of independent parts of arbitrary sizes which may be processed in any order. The number of data subsets: $N$, is possible to be specified by the user directly, or it is determined by means of an available partitioning algorithm that can regard any special application requirements. The output of the data distributor employed in the current prototype constitutes a collection of $N$ files comprising the information about the input data subsets, and $N$ job description files (written at present in JDL [29]), one file for each job-run. Additionally, the directory scheme for each separate job-run is created.

**Task Executor** manages the execution of the application, that means, it initiates all job-runs with their own input data. Its realization follows the *master-worker* strategy exploiting the *gLite* [30, 31] Grid middleware. The individual jobs are submitted to the Grid in turn using the corresponding JDL-file and input data file generated within the data distributor. The execution of the job itself is carried out by means of the generic *worker*-script which is invoked for each job-run accompanied with computed parameters.

**Output Retriever** keeps track of the execution status of individual jobs and in case of its completion it enables the retrieving of results to the user’s home. In general, this component is application specific. The Grid middleware includes mechanisms for transferring small data files needed to start a job or to check its intermediate results, however, large data files must be transferred from/to a permanent storage element with registering in a Grid File Catalogue. In order to accomplish this file transfer services must be utilized.
5 Case Study: PCNN Application

In this case study we use our framework to handle a real-world application from the image processing area. The direct image processing in multi-dimensional space is a very difficult problem, its complexity depends on dimension space of the image. Therefore, typically, the dimension of the image space is tried to be reduced into a space of significant features with a radically lower dimension which defines a new formal representation of the image. The PCNNs (Pulse Coupled Neural Networks) [32] are neural models developed for high-performance biomimetic image processing. Over the past decade, PCNNs have proved to be a suitable and efficient approach for a variety of image processing operations, including: image segmentation, feature generation, face extraction, motion detection, region growing, noise reduction, and so on.

The PCNN model involved in our test application deals with the dimension reduction of the classification space. The detailed mathematical description of the algorithm can be found in [33]. The PCNN Application Package includes the executable file, configuration file for setting model parameters, database of images, and specification file containing a list of images which are of interest. Each image is given in the form of a 2-dimensional binary matrix and is placed in a separate file. The database may cover thousands of images of arbitrary sizes. The PCNN model need to be executed over the list of images mentioned in the given specification file. There is no dependency between images, so, theoretically, all of them can be handled in parallel. In fact, however, the Data Distributor must divide the set of input images into parts including a reasonable number of images in order that the time induced by the Grid operation overhead would be amortized by the computational time of the job. For the current PCNN database consisting of 2000 images, the partitioning algorithm creates groups with at least 20 images. For each generated group of images a new specification file is constructed. The particular jobs are consecutively submitted for execution by the Task Executor; for each group of images one job-run with its own specification file passed. Within a job image files are processed sequentially in a loop. Every job-run produces a set of output files representing images in the reduced dimension space. The Output Retriever can transfer the output files to the user’s home immediately after a job has finished, or it can wait until all jobs terminate and then all output files can be packed and stored in a storage element, eventually transfered to the user.

For experiments with PCNN we used the EGEE [30] Grid infrastructure. Table 1 lists the early performance results obtained for PCNN processing the input database consisting of 1989 images. For comparison we introduce execution times of three versions: original sequential, MPI parallel, and Grid distributed. It is evident that the best performance gain is achieved by the MPI parallel version, near perfect scaling. Grid computation times (1st item) are similar as for the MPI version, however, they are increased by times (2nd item) caused by the Grid overhead.
PCNN (Input: 1989 images)

<table>
<thead>
<tr>
<th></th>
<th>Sequential version</th>
<th>MPI version</th>
<th>Grid version</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4 nodes</td>
<td>8 nodes</td>
<td>4 jobs</td>
</tr>
<tr>
<td></td>
<td>107.9</td>
<td>26.7</td>
<td>35.8 (27.2 + 8.6)</td>
</tr>
</tbody>
</table>

Table 1. Execution Times (in min)

6 Conclusion

Our early experimental results indicate that the gridifying such kind of parameter sweep applications have the potential to achieve a fairly good performance. The subject of our future work will be the development of the framework prototype based on the Web services technology and we intend to continue in the investigation of its relevance for a larger range of scientific applications.

7 Acknowledgements

This work has been supported by projects EGEE-II EU 6FP RI-031688, int.eu.grid EU 6FP RI-031857, RPEU-0029-06, RPEU-0024-06, INTAS EU No. 06-100024-9154, and VEGA No. 2/6103/6.

References


http://www.cca-forum.org/ccafe


20. GridWay: Metascheduling Technologies for the Grid.
http://www.gridway.org


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


https://www.nas.nasa.gov/Main/Features/2001/Winter/ilab.html


31. gLite: *Lightweight Middleware for Grid Computing*. http://cern.ch/glite
