Available online at www.sciencedirect.com

ScienceDirect

ELSEVIER

Future Generation Computer Systems 23 (2007) 517–522

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.com/locate/fgcs

Grid Services Base Library: A high-level, procedural application programming interface for writing Globus-based Grid services

Adam L. Bazinet[a], Daniel S. Myers[a,1], John Fuetsch[a,b,2], Michael P. Cummings[a,*]

[a] *Center for Bioinformatics and Computational Biology, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA*
[b] *Department of Mathematics and Computer Science, Pomona College, Claremont, CA 91711, USA*

Received 15 December 2005; received in revised form 26 April 2006; accepted 15 July 2006
Available online 22 August 2006

Abstract

The Grid Services Base Library (GSBL) is a procedural application programming interface (API) that abstracts many of the high-level functions performed by Globus Grid services, thus dramatically lowering the barriers to writing Grid services. The library has been extensively tested and used for computational biology research in a Globus Toolkit-based Grid system, in which no fewer than twenty Grid services written with this API are deployed.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Client/server; Parallel processing; Web-based services; Software libraries; Object-oriented programming; Distributed systems

1. Introduction

As the size and complexity of life science data has increased, so have the sophistication and computational complexity of data analysis increased. Entire data types that did not exist a relatively short time ago (e.g., complete genome sequences, large-scale microarray results, large multilocus genotypes) now constitute much of life science data. Similarly, analytical challenges including inference and combinatorial optimization have been attacked with computer-intensive methods (e.g., stochastic simulation, machine learning methods, Bayesian analysis, Markov-chain Monte Carlo sampling). As a consequence, the computational demands of life science research continue to increase. Therefore, some life science researchers are turning to Grid computing to meet their computing resource needs, following a trend toward Grid computing in academia in general [10]. However, there are several barriers to widespread use of Grid computing in the life sciences, including the lack of Grid-enabled applications and the difficulty of producing them, the deficit of Grid computing resources available for life science research, and the difficulty of using Grid computing effectively. Several of these barriers to the use of Grid computing in the life sciences are being addressed [22]. The objective of this paper is to describe middleware tools that address one specific barrier mentioned above, difficulty creating Grid-enabled applications.

Grid computing has been defined [10] as a model of distributed computing that uses geographically and administratively disparate resources. In Grid computing, individual users can access computers and data transparently, without having to consider location, operating system, account administration, and other details. In Grid computing, the details are abstracted, and the resources are virtualized.

Our ongoing research and development in Grid computing has been motivated in large part by the computational demands of our own research in computational biology and bioinformatics. This research program focuses on problems in molecular evolution and genetics, which often require approaches that are very computation-intensive. Our need for computer resources for our work led to the development of a simple Grid computing system using commodity tools [23], which was used for a large-scale simulation study [9]. Our

* Corresponding author. Tel.: +1 301 405 9903; fax: +1 301 314 1341.
E-mail addresses: pknut777@umiacs.umd.edu (A.L. Bazinet), dsmyers@mit.edu (D.S. Myers), jfuetsch@pdi.com (J. Fuetsch), mike@umiacs.umd.edu (M.P. Cummings).

[1] Present address: Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139-4307, USA.

[2] Present address: PDI/DreamWorks, 1800 Seaport Boulevard, Redwood City, CA 94063, USA.

0167-739X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2006.07.009

subsequent work has made use of the Grid middleware Globus [12] and the Berkeley Open Infrastructure for Network Computing (BOINC) [3,7], and focused on expanding the reach of Grid computing by creating a system that combines these two models (Myers, Bazinet and Cummings, in preparation). The work described here extends and complements other efforts [22] and represents our approach to making it easier to develop Grid-enabled applications using the Globus Toolkit. Although our focus is on applications used in computational biology and bioinformatics, the middleware solution we developed is general and is applicable to other domains.

The Globus Toolkit represents the current state of the art in Grid middleware. It is the focus of much of the ongoing research in Grid computing, and we can expect to see continued support and development well into the future. Based on a web services architecture, Globus provides facilities for the execution and management of jobs on remote resources, on-the-fly resource discovery, file transfer, authentication and authorization, and encryption of messages. Using the Globus Toolkit, it is possible to build large, highly-distributed, and robust computational grids.

Over the past several years, our research has been aimed at using the Globus toolkit, in combination with other Grid middleware, to create a computational Grid for scientific research. We began development with Globus Toolkit 3, which formed the backbone of our Grid system. Development continued until we had a fully functional production-level Grid system built around Globus Toolkit 3. After successful production use of this system, we focused our efforts on upgrading our infrastructure to use Globus Toolkit 4 (which was released in early 2005). The challenge of writing Grid services with the Globus Toolkit has remained constant, however, and we will now describe some of the implementation challenges we have encountered and our approaches to overcoming them.

2. The challenges of working with Globus

As might be expected in research-grade software, there are problems with the Globus Toolkit. First, the application programming interface (API) that Globus provides for writing Grid services is a relatively low-level one, and accomplishing common tasks (such as transferring a file between two systems) can often require a lot of code. Writing a fully featured application-based Grid service is not as easy as we would like it to be.

Second, Globus uses an asynchronous, event-based model for programming Grid services. Although such a model is well suited to Grid computing, where one may have to wait unknown lengths of time for operations to complete (e.g., between submitting a job and receiving the results), it is not necessarily the most intuitive programming model. In many cases the task of writing Grid services will be facilitated if it can be done using a procedural model with blocking function calls, even if the underlying infrastructure is event-based.

Third, because the Globus Toolkit is research software under continual development, there is always the possibility that the API presented to Grid services will change between versions. This is precisely what happened between Globus Toolkit 3 and Globus Toolkit 4. A perceived high probability of API change can make programmers hesitant about writing Grid services using the API.

Finally, creating a new Grid service requires creating a number of new files in a very specific directory structure and with very specific names, namespaces, and classes. This is a tedious and error-prone process at best, but one we have to repeat each time we write a Grid service. Moreover, because we were interested in having our applications run in a general framework, we designed our Grid system around the idea that every Grid-enabled application would be presented as a Grid service. Thus, we knew we would be building a significant number of services, and so it was desirable to reduce the overhead associated with this process as much as possible.

3. Our solution

To resolve the above problems, we have written the Grid Services Base Library (GSBL), which provides a high-level, procedural API for writing Grid services. In our Grid system, GSBL is the API called by our body of Grid services; at this level, no Globus code is invoked directly. Thus, in the event that the Globus API changes, only GSBL will require updating. It should also be noted that the Globus team tries to preserve concepts from version to version of the toolkit, which means that high-level GSBL-supported operations should also migrate easily. This solves the problem of a changing API; in the rest of this section, we discuss the GSBL API and how it solves the problems associated with the low-level, event-based programming model of Globus.

Admittedly, we have not attempted to provide a friendly interface to the entire Globus API or to support all possible operations. As a guiding principle of our API design we have focused on making simple and common tasks easy to implement, while leaving the programmer to the Globus API for more difficult and uncommon tasks. We note, however, that after having built twenty production Grid services for life science applications, we have yet to encounter the need to circumvent GSBL to write custom Globus code.

In keeping with standard web services procedures, we have designed our Grid system with a generalized client–service architecture in mind. As mentioned previously, each Grid service represents a Grid-enabled application (see Section 4 for a brief description of our Grid services). A remote Grid client then invokes a set of operations that cause a particular application to be run on the Grid. These operations are performed during job setup, submission, monitoring, and cleanup, and they fall into the following areas: initial configuration of Grid client–service interaction, argument processing, transfer of files between the client and the service, and submission and monitoring of Grid Resource Allocation and Management (GRAM) jobs by the service. It should be noted that GSBL is a library for writing both Grid services and Grid clients that are inter-operable in the framework outlined.

3.1. Initial configuration of client–service interaction

There are several steps that a Globus Grid client needs to take in order to establish communication with a Grid service.