APIs and End Users
Cluster and Grid Computing 2006

Urszula Herman-Izycka 1544071
Michal Ejdys 1544055
Kevin Huguenin 1603779

June 9, 2006

1 Introduction to Grid APIs

There exist quite a few different systems for low level tasks in the Grid world – a number of environments for jobs scheduling, file transfers, resource discovering is overwhelming. It complicates applications and makes porting a real nightmare. GridRPC (see 3.2.1) regards [11] programming directly on top of Globus I/O as performing parallel programming using only the Linux API on a cluster. And that is where high level APIs come with rescue.

2 End Users

End users are omnipresent throughout the production phase [1]: GRID designers have to figure out who they are, their needs and their expectations so as to build well adapted and efficient user interfaces (APIs, GUIs, WebPortals, ...). Moreover, designing a grid – especially its interfaces – depends on whether the environment is a campus testbed or a national production infrastructure.

2.1 Who are they?

The final goal of grid designers is to provide computational or data storage grids as easy to use as the electricity power grid. At first sight everyone is a potential grid end user, but at this point the grid end users are mainly researchers, students and engineers.

2.2 What do they expect from the GRID?

End users basically expects simple access to the grid, easy job creation and submission and high performances [2].

The first point implies fast and easy protocols for registration, authentication and authorization thanks to user-friendly interfaces for instance ([3]).

Job creation and submission should also be done through easy-to-use interfaces which provide a feature rich environment and hides the heterogeneity of the underlying grid. Concretely, end users expect a minimal set of interfaces – such as APIs (see 3) or graphical interfaces for dynamic composition of reusable components [3] – that focus on functionality rather than technicalities to build their applications and user-friendly interfaces to visualize and control the execution environment.

2.3 Campus Grids v.s. Production Grid Services

Campus grids [4] aims at emphasizing support to local researchers, educators and students. These grids are testbeds, thus failure resilience is not the highest priority. The authentication policy is basic: one could login to the grid if and only if he has an account on a cluster. Thus security issues are minor and there are no commercial issues. Submitting a job basically consists in running a program on a cluster. The well known DAS-2 Grid is an example of a campus grid.

On the other hand, Production Grids [4] aim at providing infrastructure and services – which actually produce reliable results – to individuals and communities. This kind of grids raises much more issues than campus grids, such as security issues, failure resilience, efficiency, scalability, geographical dissemination, commercial issues, ...

3 Specialized Grid APIs

3.1 DRMAA

Main goal of DRMAA (Distributed Resource Management Application API, [5, 6, 7]) was to prepare one high-level API for many different DRM (Distributed Resource Management) systems, e.g. SGE, Condor, GRAM. From the very beginning of work on DRMAA, work group closely cooperated with DRMs developers as well as with SAGA (see 4.3) work group. DRMAA aims at preparing API offering access to DRMs from users perspective. Users want to easily perform most common tasks, rather than be forced to do sequences of several small steps.

DRMAA's currently supports Sun Grid Engine (in C and Java), Condor (in C), Torque (in C) and GridWay (in C). Furthermore, it offers language bindings to Perl and Python.

3.2 RPC for the Grid

Remote Procedure Call is a well known and very commonly used way of network programming. There were several attempts to port this concept to the Grid environment. What makes Grids special in the context of RPC programming [11] is:

- dynamic resource discovery and scheduling,
- scientific datatypes (large matrices),
- big variety of calls complexity (from a second to days or weeks).

There exist several implementations of RPC for Grid, e.g. NetSolve (see 3.2.2, [8]), Ninf [9] and DIET [10]. All of them provide the same functionality, nonetheless they are not interoperable. For that reason, the need for standardization arisen, resulting in creating GridRPC API.

3.2.1 GridRPC

GridRPC [11, 12] offers API for function calls, execution control and synchronization. It hides service discovery behind simple API call, i.e. obtaining function handle.

3.2.2 NetSolve

NetSolve [8] (product of University of Tennessee) is a very interesting implementation of the RPC for the Grid concept. Special agents are used to choose the best server that offers functionality requested by the user, basing on current load of servers. The NetSolve RPC library takes care of contacting agent, acquiring available server list (ordered with respect to their load) and contacting one of them.

NetSolve has been modified to implement GridRPC API.

3.3 Grid Checkpoint Recovery API

In unstable Grid world, where nodes go up and down and network connections fail, many long running application do checkpoiting for failure recovery. Checkpointing helps also in migrating processes between Grid resources, parameter sweeping, monitoring and visualization.

For that functionality to be standardized, works on GridCPR API [13, 14] have been started. However, its workgroup after a few meetings decided to integrate concept with a much more bigger effort – SAGA (see 4.3).

4 Towards standardized API

Currently, there are many APIs for running programs on the Grid. Some of them are targeted at special groups of researchers, whereas others aim at solving industry problems. Therefore it is practically impossible to write an everywhere running application, which can be used by different groups of users. A one standard API, like an MPI one, would solve such problems and the need for that is constantly growing. With that recognized the work on SAGA API is conducted. The researchers analyze the use cases as well as other current systems (e.g. RealityGrid, CoG Kit and GAT).

4.1 RealityGrid

RealityGrid provides environment for writing programs, where evolution of computation can be controlled in real time. The output of the computation can be analyzed and possibly visualized on the fly, which proves very useful when solving scientific modelling problems. Programs solving such problems profit enormously from being run on the Grid, as Grids provide different resources ,,in one place". Thus some specialized computers take care of simulation, whereas others of visualization.

4.2 Commodity Grid Kit

CoG Kit is a high level framework for grid programming with Java and Python bindings. It has been developed since 1997 and therefore has been successfully used in many projects, such as LIGO or LEAD.

CoG Kits architecture is based on the layered module concept. CoG Kit provides API with advanced abstractions for job execution, file transfers, workflows and job queues. Important feature is that CoG Kit can be run on different grid middleware, with dynamic class loading depending on the existing production grid. This idea resembles GAT architecture. CoG Kit developers recognized the need for a user friendly environment. Therefore the system provides different methods for workflow definitions, the main one being an XML based Karajan engine, as well as prototypes for user centric Desktop workspace or experiment management tool.

4.3 SAGA

SAGA (Simple API for Grid Applications) is one of the latest efforts in grid community to finally provide a more standardized API. It can be thought of as a grand unification of the previous efforts and systems. The user should get a high level API with bindings for most popular languages.

The important design feature of SAGA is its 80-20 Rule: ,,Design an API with 20% effort and serve 80% of the application use cases". Therefore a lot of work during the first stages of development has been devoted to the use case analysis, as well as the experience analysis of former projects. In SAGA object-oriented approach is used and there are different packages responsible for different areas of grid subsystems (session handling and security, jobs and tasks, data management, steering and monitoring, task dependencies, GridRPC).

References

End Users, Campus Grids and Production Grid Services

[1] US DoD (Department of Defense) HPCMP (High Performance Computing Modernization Program)

[2] Frank Wrthwein, *Me My friends the Grid*, UCSD (University of California, San Diego)

[3] M. Baker, R. Buyya and D. Laforenza, *The Grid: International Efforts in Global Computing*

[4] I. Foster, *The Grid: From Lab to Practice*, Argonne National Laboratory: University of Chicago

DRMAA

[5] http://www.drmaa.org/ – DRMAA's homepage

[6] https://forge.gridforum.org/projects/drmaa-wg – DRMAA's work group homepage

[7] *DRMAA Tutorial C and Java Language Bindings*, GGF 12

GridRPC

[8] http://icl.cs.utk.edu/netsolve/ – NetSolve's homepage

[9] http://ninf.apgrid.org/ – Ninf's homepage

[10] http://graal.ens-lyon.fr/ diet/ – DIET's homepage

[11] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee and H. Casanova, *GridRPC: A Remote Procedure Call API for Grid Computing*, July 2002

[12] https://forge.gridforum.org/projects/gridrpc-wg/ – GridRPC's work group homepage

GridCPR

[13] http://gridcpr.psc.edu/GGF-/ - GridCPR's homepage

[14] https://forge.gridforum.org/projects/gridcpr-wg – GridCPR's work group homepage

RealityGrid

[15] http://www.realitygrid.org/

[16] http://www.sve.man.ac.uk/Research/AtoZ/RealityGrid/

CoG Kit

[17] http://wiki.cogkit.org/index.php/Main_Page

SAGA

[18] https://forge.gridforum.org/projects/saga-rg/

[19] www.cs.vu.nl/ kielmann/papers/saga-sc05.pdf