Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

A conceptual model for grid-adaptivity of HPC applications and its logical implementation with components technology.

Alberto Machì, Saverio Lombardo,

RT-ICAR-PA-03-13 **dicembre 2003**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sezione di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

A conceptual model for grid-adaptivity of HPC applications and its logical implementation with components technology

Alberto Machì, Saverio Lombardo,

Deliverable Progetto Grid.it WP9 Librerie Scientifiche

Rapporto Tecnico N.: **RT-ICAR-PA-03-13**

Data: **dicembre 2003**

[1,2] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sezione di Palermo .

Abstract.

Today grid middleware is complex to be used, the development of grid-aware applications is error-prone and the Virtual Organization grid paradigm contrasts with traditional HPC optimisation strategies based on resource stability and known cost models of inter-node interaction patterns.

The authors analyse several aspects of grid adaptivity, and identify 5 roles: the active resource/execution manager, the proactive resource administrator, the reactive quality-service coordinator, the passive resource coordinator .

They present a hierarchical model for a component-based grid-software infrastructure in which the resource administrator and the resource coordinator roles are assigned to grid middleware and the quality-service role to HPC skeletons. Component interfaces and interactions are described.

The resource administrator mimics functionalities of components containers of service-oriented architectures. The resource coordinator manages the life cycle of sets of processes on top of a pool of grid resources and offers to the upper layers a Virtual Private Grid facade similar to a standard processor cluster facility.

1. Introduction

The computational Grid paradigm defines a flexible, secure, coordinated large-scale resource-sharing model. Its focus is on large-scale problem solving in dynamic, multi-institutional virtual organizations [1].

High performance computing has been, instead, traditionally oriented to performance optimisation of proprietary resources on local or wide area networks, based on exploiting knowledge of management policies at any level (computational models, resource connection patterns, cost models of the processor interaction graphs). In particular, the structured parallel programming approach has embodied such knowledge into patterns for the management of set of processes described by notable Directed Graphs, called *skeletons* and *parmods* [2-3].

Skeletons are automatically coded by parallel compilers to keep high the parallel efficiency while maintaining low parallel programming difficulty and software portability.

Code developed with such a structured approach for environments mapped on static networks of resources, and controlled by stable policies of exclusive resource allocation or partitioning, is inefficient on to grid environments maintained as Virtual Organizations on wide-area networks of dynamically discoverable and shareable resources is ineffective because of the resource unreliability concept intrinsic in the Virtual Organization model.

One approach to such a problem is to develop self-adaptive parallel coordination patterns, in which some node in the process graph is assigned to maintain awareness of past grid nodes performance statistics and grid resources present status to optimally adapt component coordination pattern. For instance, grid-awareness can be used to drive load balancing through optimal mapping of virtual processes over physical nodes according to effective node performance [4]. Otherwise it can be used for substituting faulty nodes in a process graph or to redistribute data stream processing among workers of a farm if any worker is not honouring its performance contract. [5].

Programming adaptivity for each coordination pattern is a complex and error-prone activity and makes porting of legacy (structured) parallel code hard.

We propose to adopt a hierarchical programming approach in which grid adaptivity is distributed among various layers of the software environment playing different roles: the execution environment (active resource/execution manager), the application coordination middleware (proactive resource administrator), the application component layer (reactive quality-service coordinator) and the platform management middleware (passive platform coordinator).

Specifically, adaptivity is factorised into a) discovery and reservation of grid nodes and services and definition of application virtual processes graph assigned to the execution environment b) optimal mapping of application process graph assigned to the application coordination middleware; c) load balancing for nodes of the actual instance of the process

graph assigned to application component(s); d) monitoring of graph process set and (re) configuration of process ports, assigned to the lower middleware layer.

Two patterns are defined: the *Virtual Private Grid (VPG)*, a passive facade pattern which hides management of physical nodes and processes on the grid, and the *Grid-aware Component Administrator*, a reactive pattern which hides actual management of application components according to Quality of Service criteria.

The remainder of this paper is organized as follows. Section 2 sketches the hierarchical adaptivity model. Section 3 describes the functionalities of logical components implementing the model. Section 4 describes current status of prototypes implementation of the *Virtual Private Grid* runtime support layer.

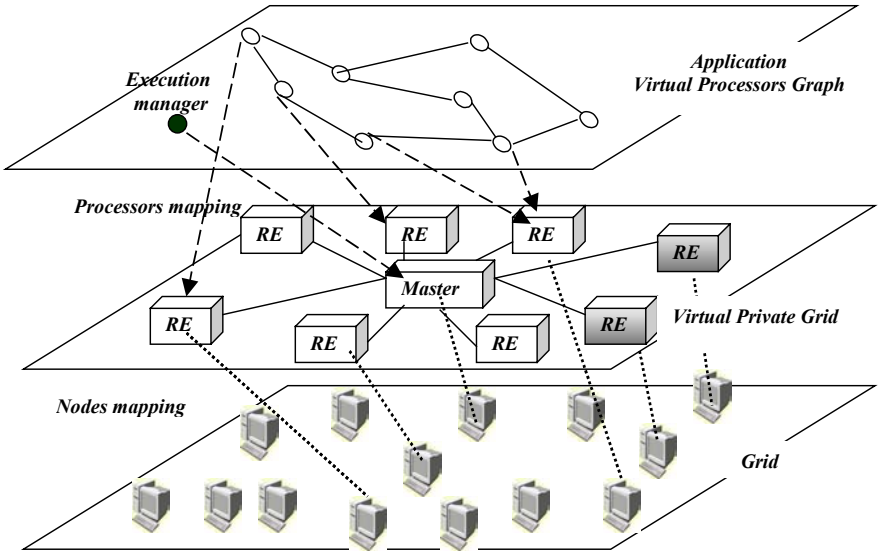

Fig. 1. Hierarchical representation of an HPC application over the grid. *Upper layer*: HPC application as a coordination of Virtual Processors DVG and an external Process Manager. *Intermediate layer*: Virtual Private Grid of computational resources administrated by middleware on behalf of application to support adaptivity: *DPG* + spare nodes (dashed). *Lower level*: actual temporary mapping of VPG nodes over the grid.

2. A hierarchical conceptual model for grid adaptivity.

Let's represent an HPC application at conceptual level as a directed graph DVG describing interaction of virtual processes and at the implementation level with graph DPG of the physical processing units that implement them.

Each graph node describes some defined application functionality or resource (a data or event source/sink, a computation, a control, a service), each arc describes a data /event transfer or a path. DVG and DPG nodes are assigned weights representing respectively their expected processing load and effective available power. DVG and DPG arcs are assigned weights representing respectively expected transfer throughput and effective available bandwidth.

After virtual process coding and *DVG* graph composition, application life-cycle can be factored into five main phases: a) *discovery* and reservation of a meaningful set of proper physical resources, b) *mapping* of *DVG* into a *DPG*, c) *initialisation* of *DPG* and *activation* plus execution of *DVG* processes on *DPG*, *release* of resources allocated at completion.

HPC practice requires *DVG* execution to satisfy Quality of Service constraints sometimes express as a *performance contract* to be honoured by the platform enabling *DVG* operation. Adaptivity of the HPC application is required to cope with *events* affecting the composition of the *DVG* resource set or even the behaviour of any of its elements:

- Fault of a processing node (graph disconnection)
- Fault of a connection between nodes (graph disconnection).
- Availability of additional resource (sub graph join)
- Redirection of a link towards an external service (leaf node cut+leaf node join)
- Change of node effective power or link bandwidth (node/arc weight change)
- Other events are best represented as overall processing states:
- Performance under contract range (insufficient node weights or sub optimal DVG to DPG mapping)
- Parallel inefficiency (excessive node weights, or sub optimal DVG mapping).

Adaptation to such events is possible only if proper constraints are satisfied:

- Graph disconnection is a catastrophic event whose recovery from checkpoints is possible only if a mechanism for periodical backup of application status at checkpoint is expressly provided by application and run-time support [6].
- Adaptation to worker node fault or to node join is possible in master-slave implementations of skeletons like a *map* or a *farm* if a mechanism is provided to force any graph node to perform synchronous re-initialisation of the communication environment.
- Adaptation to dynamic redirection of a link to a leaf node implementing a service requires asynchronously forcing of the node to close its channel to the service and to open a new one.

- Adaptation to variance of worker node effective power or link bandwidth is possible for stream-parallel skeletons if knowledge of worker sub graph weights is expressly used for workload partitioning in the implementation template [7].
- Finally, adaptation to global performance states requires access to resource management privileges, normally reserved to a coordination process external to the process graph.

Each adaptivity case requires a different level of activity and grid-awareness. It can be modelled using different actors playing hierarchically cooperative roles.

At top hierarchy level we find (re) selection of proper resources (nodes, services and DVGs). It requires grid discovery ability, detailed grid-awareness, reservation privileges and an adequate policy to coordinate resource provision and application quality of service. These tasks define the role of an *active resource/execution manager*.

At intermediate level we lay adaptive management of available grid resources for optimal execution of a defined application graph of processes. Taking advantage of self optimisation capability embodied in parallel skeleton templates, grid-adaptivity may be factorised in two roles: optimal administration of a pool of resources on behalf of a quality-application, and optimal administration of the set of resources assigned to a single application.
The first role requires definition of each application quality in terms of a performance contract, selection of optimal subset for DVG to DPG mapping, monitoring of DPG performance and a policy for DPG reconfiguration. These tasks define a *proactive resource administrator* driven by a moderately complex ontology.
The second role mimics load (re) balancing of physical processes over a cluster of virtually privates inhomogeneous resources labelled with their effective quality indexes plus partial reconfiguration of the processor graph after in the event of their variations. These tasks define *reactive quality-service coordinator* implemented in some parallel skeletons.

At lowest level we lay effective monitoring of resource status, support for DVG to DPG mapping and re-mapping, detection and registration of events requiring attention and possible adaptation, tasks executable by a passive *resource coordinator.*

The hierarchical role model for HPC grid-adaptivity may be mapped to a component-based grid software infrastructure. The resource administrator and the resource coordinator roles are assigned to grid middleware while the quality-service coordinator role is assigned to skeletons. The *resource administrator* mimics functionalities of components containers of service-oriented architectures. The *resource coordinator* manages the life cycle of sets of processes on top of a pool of grid resources and offers to the upper layers a *Virtual Private Grid* facade similar to a standard processor cluster facility.

Next section describes a logical view of the software infrastructure as cooperation as the logical interaction among software components playing model roles.

3. Functional model of middleware infrastructure

Software component technology is a young programming-paradigm, even though its definition is quite old. Its aim is to enable the development of applications by composing existing software elements in an easy way mode. Among various definitions of the component concept, we report Szyperski's one [8]: "*A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties*".

Several components models have been proposed. The Globus Project has proposed the OGSA architecture for grid services and a component architecture for adaptive grid programming compliant with OGSA been defined in [1]. A component architecture focusing on HPC grid programming is presently being developed by the *Grid.it* project [9].

Grid.it components expose their functionalities through a series of interfaces belonging to several classes (RPC, streams, events, configuration). Interface signature together with implementation technology and communication protocol defines a *port type*. Components with same or compatible *port-types* can be connected together [10]. In the framework of the Grid.it project, we exploit this component architecture to implement a graceful distribution of adaptivity roles events and actions. Figure 2 shows the components implementing the architecture and their interactions. Each component is represented as an UML-package and its interfaces as UML-classes [11].

Passive roles provide slave functionalities through *p_ports* (factory, service provide and config), active roles use them through RPC *u_ports* (discovery, system, service_invoke) as in CCA compliant frameworks [15]. Event ports of the run time support provide to the reactive application component an event bus for meaningful events registration and notification, to enable its reactive role.

The Execution Environment uses services exposed by Grid services, Component administrator and Application components.

The Component Administrator component exposes the following interfaces:

- **component_factory**: an extension of the factory design pattern [12] to the domain of distributed computing. It has the same goal of the OGSA Factory Port Type, even though it differs in some details. Factory services include submission of a Virtual Process Graph with its QoS profile and VPG hardware resources creation and modification.
- **service_provide:** it exposes a set of functionalities about the status of submitted applications.
- **component_config:** modification of leaf nodes of DVG (external services binding).

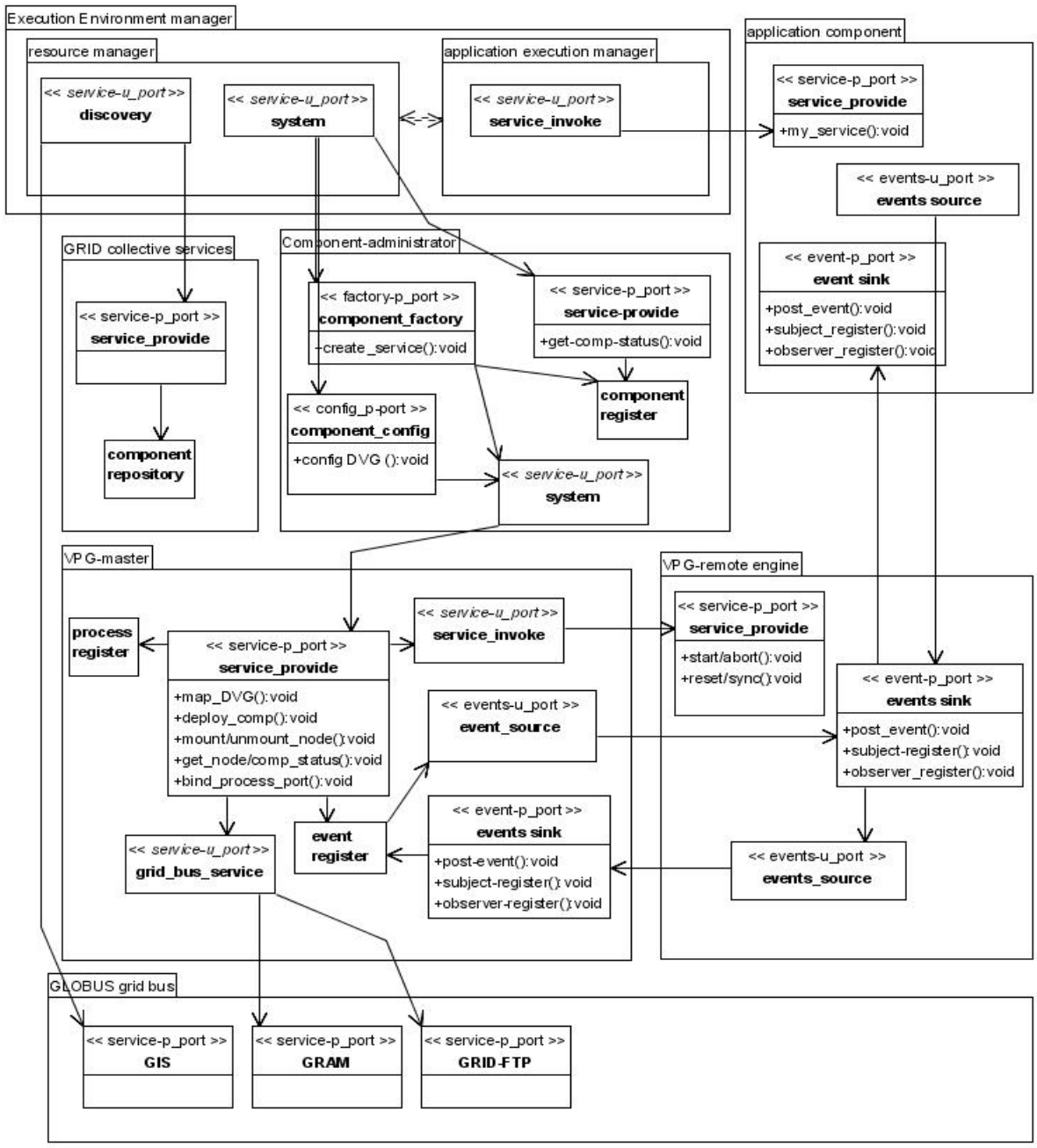

Fig. 2. Grid Layered Components Framework

The Component Administrator uses services of VPG master **service_provide** port to:

- deploy a set of processes (DVG) with related libraries on the VPG;
- start a set of processes (DPG) on the VPG;
- retrieve information about the status of managed hosts (nodes of the VPG) and about life status of started processes.
- retrieve information about the proceeding status of DPG processes to detect QoS violations;
- send message to application-components to notify that a self-configuration is needed, if possible.

4. Current Implementation of the VPG RunTime Support

A VPG Runtime Support prototype is presently being developed as a research activity of Project GRID.it.

Actual component implementation is based on usage of several design patterns [13]: acceptor-connector, reactor, proxy, wrapper and adapter. A platform independent SDK for these patterns is provided by open-source object-oriented framework ACE [14], which enables code portability of the run-time system.

The *Virtual Private Grid* pattern is implemented by the following two components:

1. **VPG-Master**: the VPG front-end. It administers hosts by exposing methods for administrating node facilities (mount, un-mount, keep alive, get-status) and for controlling set of processes (deploy, start, kill, delete, get-status). It exposes this functionality by accepting XML-commands through a socket service-provide port.
2. **VPG-Remote Engine**: a daemon running on each host mounted on VPG as a slave for VPG-Master requests. It implements the remote run-time environment, administering, under master control, local processes lifecycle (run, kill, status, clean) and redirects events between VPG master and Application components.

The Master communicates with each Remote Engine in two ways: by ports-connection to invoke control of process lifecycle, and by event notification to delivery component to component event messages.

Grid nodes management and file transfer is implemented over the Globus toolkit2 services: GRAM (for start-up of the Remote Engine), GridFTP for deploying processes DPG, GSI protocols for authorization and secure file transfer.

Acknowledgements

Work supported by Italian MIUR Project FIRB-Grid.it,
Workpackage 9 Scientific Libraries
Workpackage 8: Programming Environments

References

[1] F.Berman, G.C. Fox, A.J.G.Hey: Grid Computing. Making the Global Infrastructure a Reality. Wiley 2003

[2] J. Darlington, Y. Guo, H. W. To, J. Yang, *Parallel skeletons for structured composition*, In Proc. of the 5th ACM/SIGPLAN Symposium on Principles and Practice of Paral1el Programming, Santa Barbara, California, July 1995, *SIGPLAN Notices* 30(8),19-28.G.

[3] M. Vanneschi: The programming model of ASSIST, an environment for parallel and distributed portable applications. Parallel Computing 28(12): 1709-1732 (2002)

[4] Zizhong Chen, Jack Dongarra, Piotr Luszczek, and Kenneth Roche"Self Adapting Software for Numerical Linear Algebra and LAPACK for Clusters www.cs.utk.edu/~luszczek/ articles/lfc-parcomp.pdf

[5] C. Ferrari, Concettina Guerra, G. Canotti "A grid-aware approach to protein structure comparison" Journal of Parallel and Distributed Computing Vol 63 Issue 7-8 July 2003 pp. 728-737.

[6] P.D'Ambra, M.Danelutto, D. di Serafino, M.Lapegna:"Integrating MPI-based numerical software into an advanced parallel computing environment".Proc. of the 11th EUROMICRO Conf. on Parallel, Distributed and Netwok-based Processing, IEEE Publ., 2003, pp.283-291.

[7] A. Machì, F. Collura "Skeleton di componenti paralleli riconfigurabili su griglia computazionale map e farm ".TR ICAR-PA-12-03 - Dec 2003.

[8] Szypeski, C., Component Software: Beyond Object-Oriented Programming, Addison-Wesley, 1998.

[9] M. Vanneschi "Grid.it : a National Italian Project on Enabling Platforms for High-performance Computational Grids" GGF International Grid Summer School on Grid Computing Vico Equense Italy July 2003 www.dma.unina.it/~murli/SummerSchool/session-14.htm.

[10] M. Aldinucci, M. Coppola, M. Danelutto, M. Vanneschi, C. Zoccolo "Grid.it component model "Project Grid.it WP8 Deliverable, Jan 2004.

[11] J. Rumbaugh, I. Jacobson, G. Booch "The Unified Modeling Language Reference Manual" Addison Wesley 1998.

[12] E. Gamma, R. Helm, R. Joyhnson, J. Vlissides **"**Design Patterns . Elements of Reusable Object-Oriented Software". Addison-Wesley.

[13] Douglas C. Schmidt, Michael Stal, Hans Rohnert and Frank Buschmann "Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects" Wiley & Sons in 2000, ISBN 0-471-60695-2.

[14] The Adaptive Communication Environmenthttp://www.cs.wustl.edu/~schmidt/ACE.html

[15] Rob Armstrom, Dennis Gannon, Katarezyna Keahey, Scott Kohn, Lois McInnes, Steve Parker, and Brent Smolinsk. "*Toward a common component architecture for high-performance scientific computing*". In Conference on High Performance Distributed Computing, 1999 [10] The Common Component Architecture Technical Specification – Version 0.5. http://cca-forum.org/bindings/old-0.5/.