A Supercomputing API for the Grid[*]

Károly Bósa[†] and Wolfgang Schreiner[†]

1. Introduction

No application can execute efficiently on the grid that is not aware of the fact that it runs in an heterogeneous network environment with heterogeneous nodes. We report on an ongoing work whose goal is to develop a distributed software framework and an API for grid computing which shall empower applications to perform scheduling decisions on their own and utilizing the information about the grid environment in order to adapt their algorithmic structure to the particular situation. Since our solution hides low-level grid-related execution details from the application by providing an abstract execution model, it is able to eliminate some algorithmic challenges of nowadays grid programming.

Regard an example where a user intends to execute a tree-like multilevel parallel application on the grid. She specifies in advance that the given application should consist of 20 processes organized into a 3-levels tree structure. On the lowest level leaves belonging to the same parent process should form groups such that each group contains at least 5 processes scheduled to the same local network environment. For this specification, our software framework is able to determinate a (nearly) optimal distribution of processes on the momentarily available grid resources and to start the processes according to this distribution. Furthermore, our API is able to apply at runtime a corresponding mapping between the predefined roles of processes in the specified hierarchy (global manager, local manager and workers) and the allocated pool of grid nodes such that it minimizes the execution time.

2. Architecture

In our approach, the user assigns to each given parallel program a pre-defined *schema* that specifies a preferred communication pattern of the program in heterogeneous network environments. In our system the following kinds of communication schemas are currently employed [1]: the schema *singleton* specifies a number of processes which should be scheduled to the same local network environment; the schema *groups* specifies the number of processes and either the accurate size of the *local groups* (the number of processes in the same local network environment) or a minimum size for the local groups and some restriction for the number of the local groups; the schema *graph* is similar, but it additionally defines edges/links between the local groups such that they describe a communication pattern; the schema *tree* specifies a tree-like multilevel parallelism with the given number of processes and the given number of tree levels, such that the sizes of the local groups located on the lowest level (the level of leaves) are not determined but some restrictions are given for it; last but not least the schema *ring* is similar to the schema graph, but the local groups always compose a ring.

[*]The work described in this paper is supported by the Austrian Grid Project, funded by the Austrian BMBWK (Federal Ministry for Education, Science and Culture) under contract GZ BMWF-10.220/0002-II/10/2007.

[†]Research Institute for Symbolic Computation (RISC), Johannes Kepler University,
email: Karoly.Bosa@risc.uni-linz.ac.at, Wolfgang.Schreiner@risc.uni-linz.ac.at

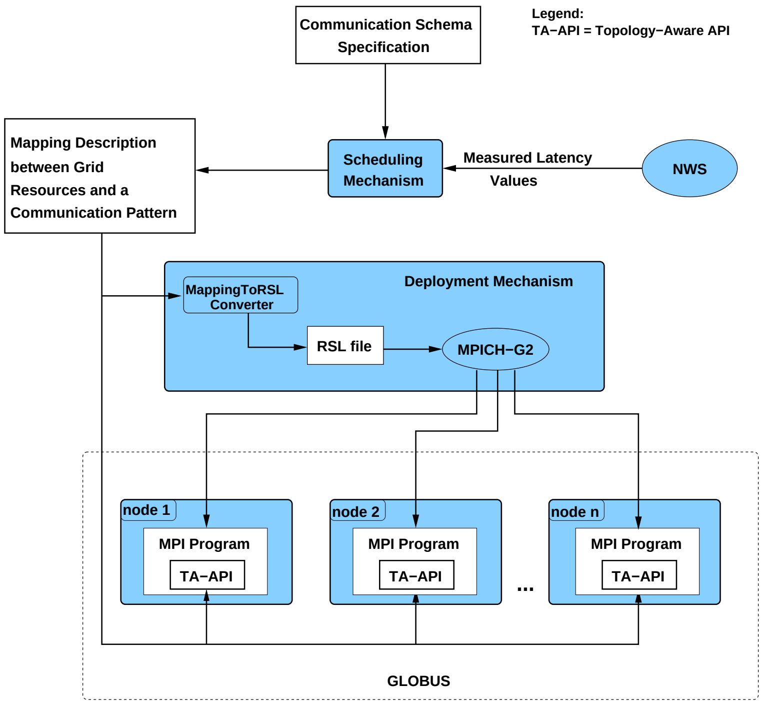

Figure 1. Overview on the Software Framework

We have already designed and implemented the first prototype version [1, 2] of our software framework which is based on the pre-Web Service architecture of the Globus Toolkit and MPICH-G2 [3]. Our solution consists of three major components (see Figure 1):

Scheduling Mechanism depends on the *Network Weather Service (NWS)* [6], which became a de facto standard in the grid community as a performance prediction tool. Since the NWS provides all necessary information concerning the utilizable grid resources, namely the list of the available hosts, a forecast for the available CPU fractions on these hosts and a forecast for the latency values in milliseconds are predicted for each pair of hosts, the user needs therefore not know any detail of the grid architecture. Of course, in addition to these performance characteristics the scheduling algorithm needs a preferred communication pattern of a particular application the user must specify in an XML format.

Before each execution of a parallel program on the grid, the Scheduling Mechanism adapts and maps a preferred communication pattern of the program to the available grid resources such that it heuristically minimizes the assessed execution time. It works roughly as follows:

1. First we classify all the links between each pair of hosts according to the order of magnitude of latencies. For the generated classes we assign an ascending sequence of integer numbers (*latency levels*). To the class which comprises the fastest links we assign the level 1, to the next one we assign the level 2 and so forth.
2. We compose some not necessary disjoint clusters (let us call them *latency clusters*) from all the given hosts such that the latency levels of the links between any two member hosts of such a cluster cannot exceed a certain value (some of these latency clusters may comprise some others with less maximum latency level). Furthermore each host itself is regarded as a latency cluster with the latency level 0. The generated latency clusters are stored in a list which is sorted according to the max. latency levels in an ascending order.

3. We generate all those *partitioning* of processes (in which processes are organized into various local groups) which fulfil the given preferred communication pattern of a program.
4. Finally we map the generated process partitions to some latency clusters according to some heuristic (which helps to avoid the combinatorial explosion of possibilities) and find a reasonably efficient scheduling for the program. In the comparisons of the mappings the algorithm takes into consideration the following characteristics: the maximum latency level within the local groups, the absolute maximum latency level in the entire mapping, the number of the local groups and the average latency value among the local groups.

The output of the algorithm is an XML-based *mapping file* (describing a mapping between the grid resources and the given communication pattern).

Deployment Mechanism is based on the starting mechanism of the grid-enabled MPI implementation MPICH-G2 [3]. It distributes the mapping file on the corresponding grid nodes and starts the processes of the given program on the grid according to the mapping file.

Topology-Aware API is an addition to the MPICH programming library. Its purpose is to query mapping files and inform parallel programs how their processes are assigned to some physical grid resources and which are the designated roles for these processes, such as: in which local group a particular process is involved; which are the characteristics of local groups, of graphs (e.g.: neighborhoods of a group, distance of two groups, etc.), of trees (e.g.: depth of a tree, parent and children of a process, etc.) or of rings.

For representing the versatility of our API, we have developed a simple distributed example application [2] which can be used to establish different kinds of the tree-like multilevel parallelism on the grid according to a mapping file.

Our implemented supercomputing API and the Deployment Mechanism have already been tested successfully on the grid sites `altix1.jku.austriangrid.at` (Altix 350) and `lilli.edvz.uni-linz.ac.at` (Altix 4700). In the final version of the paper, we are going to report on the full functionality of our completely implemented software framework.

3. Related Work and Conclusions

Obtaining high-performance on the grid requires a balance of computation and communication among all involved resources. Currently this can be done by manually managing computations, communications and data locality using message-passing (e.g.: MPI) or remote procedure call (e.g.: GridRPC). Although GridRPC [4] may become an OGF standard as a parallel programming interface for the grid, but it is still based only on the Client-Server model (as any other remote procedure call APIs) and lacks the versatility and power of message-passing based APIs.

While MPI addresses some of the challenges in high-performance grid computing, it was originally designed only for clusters or other homogeneous network environments. A parallel programming environment evolved for the grid must be *topology-aware* in that sense that it must be aware of and exploit the characteristic of an available physical network architecture. Typical topology-aware programming tools are e.g. *MPICH-G2* [3] and *MPICH-VMI* [5]. Both of them are grid-enabled MPI implementations based on the MPICH library. MPICH-G2 uses some grid services provided by the Globus Toolkit pre-Web Service architecture. MPICH-VMI utilizes the middleware communication layer *Virtual Machine Interface (VMI).*

Summarizing their achievements, we can say that existing topology-aware programming tools make available the given topology information on the level of their programming API and they optimize (only) the collective communication operations (e.g.: broadcast) with the help of the topology information such that they minimize the usage of the slow communication channels. But they are still not able to adapt the point-to-point communication pattern of a parallel programs to network topologies such that they achieve a nearly optimal execution time on the grid. Compared to these existing topology-aware programming tools, the major advantages of our solution are the following:

- It takes into consideration the point-to-point pattern of a MPI parallel program and tries to fit it to a heterogeneous grid network architecture,
- It preserves the achievements of the already existing topology-aware programming tools. This means the topology-aware collective operations of MPICH-G2 are still available, since MPICH-G2 serves as a basis for our software framework.
- Our system eliminates the algorithmic challenges of the high-performance programming on the dynamic and heterogeneous grid environments. Programmers need to deal only with the particular problems which they are going to solve (like in a homogeneous cluster environments).
- The distribution of the processes is always conformed to the loading of the network resources.

In the final version of the paper, we intend to present some comparative benchmarks between the pure MPICH-G2 framework and our topology-aware solution.

References

[1] Karoly Bosa and Wolfgang Schreiner. Initial Design of a Distributed Supercomputing API for the Grid. Austrian Grid Deliverable AG-D4-2-2008_1, Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, Austria, September 2008.

[2] Karoly Bosa and Wolfgang Schreiner. A Prototype Implementation of a Distributed Supercomputing API for the Grid. Austrian Grid Deliverable AG-D4-1-2009_1, Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, Austria, March 2009.

[3] N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing (JPDC)*, 63(5):551–563, May 2003.

[4] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and Casanova. A GridRPC Model and API for End-User Applications. GridRPC Working Group of Global Grid Forum, June 2007.

[5] A. Pant and H. Jafri. Communicating Efficiently on Cluster Based Grids with MPICH-VMI. In *CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 23–33, Washington, DC, USA, 2004. IEEE Computer Society.

[6] Rich Wolski, Neil T. Spring, and Jim Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.