Austrian Grid

Report on the First Feature-Complete Prototype of a Distributed Supercomputing API for the Grid

Document Identifier:	**AG-D4-1-2010_1.pdf**
Status:	**Public**
Workpackage:	**4**
Partners:	**Research Institute for Symbolic Computation (RISC)**
Lead Partner:	**RISC**
WP Leaders:	**Wolfgang Schreiner (RISC)**

Delivery Slip

	Name	Partner	Date	Signature
From				
Verified by				
Approved by				

Document Log

Version	Date	Summery of changes	Author
1	31.03.2010	Initial Version	K. Bosa, W. Schreiner, F. Priewasser

Report on the First Feature-Complete Prototype of a Distributed Supercomputing API for the Grid

Karoly Bosa
Wolfgang Schreiner
Friedrich Priewasser

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University Linz
{Karoly.Bosa, Wolfgang.Schreiner,
Friedrich.Priewasser}@risc.uni-linz.ac.at

March 29, 2010

Report on the First Feature-Complete Prototype of a Distributed Supercomputing API for the Grid

Károly Bósa Wolfgang Schreiner Friedrich Priewasser

March 29, 2010

Abstract

In the frame of the Austrian Grid Phase 2, we have designed and implemented an API for grid computing that can be used for developing grid-distributed parallel programs without leaving the level of the language in which the core application is written. Our software framework is able to utilize the information about heterogeneous grid environments in order to adapt the algorithmic structure of parallel programs 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. In this paper, we report on the first feature-complete prototype of our topology-aware software system extended with some benchmark results and a performance comparison between MPICH-G2 and our system.

1 Introduction

In this paper, we report on a completed work whose goal was to design and develop distributed programming software framework and API for grid computing [5]. This software system is able to utilize the information about the grid environment in order to adapt their algorithmic structures to the particular situation.

Our solution is an advanced topology-aware programming tool which takes into account not only the topology of the available grid resources but also the point-to-point communication structure of parallel programs. In our approach, a pre-defined *schema* is assigned to each given parallel program

that specifies preferred communication patterns of the program in heterogeneous network environments. The execution engine first adapts and maps this schema to the currently available grid resources and then starts according to this mapping the processes on the grid. Our API contains function calls which are able to query all the details of the mapping information which contains both the adapted communication structure of the program and the topological information of the allocated grid resources.

Regard an example where a user intends to execute a tree-like multi-level parallel application on the grid. She specifies in advance that the given application shall consist of 20 processes organized into a 3-levels tree structure. On the lowest level leaves belonging to the same parent process shall 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 suitable partition of processes on the currently available grid resources and to start the processes according to this scheduling. The partition is based on some heuristics, e.g.: our framework prefers such tree structures where the sizes of the groups formed by the leaf processes belonging to the same parents are maximal; consequently the processes of each such group can be scheduled to a cluster. Furthermore, our API maps at runtime the predefined roles of processes in the specified logical hierarchy (global manager, local manager and workers) to the allocated pool of grid nodes such that the execution time is minimized.

The rest of the paper is organized as follows. First we give in Section 2 an overview on the overall software architecture of our grid programming framework and report the implementation state. In Section 3 we discuss the applied scheduling algorithm in detail whose implementation was completely finished in the last project phase. Finally, we present in Section 4 a performance comparison with benchmark results between MPICH-G2 and our system and conclude in Section 5.

2 The Overall Software Architecture

We have finished the implementation of a first feature-complete prototype version of our software framework [5] called *"Topology-Aware API for the Grid"* (*TAAG*). The system is based on the pre-Web Service architecture of the Globus Toolkit [1] and on MPICH-G2 [9] and it consists of three major components (see Figure 1):

Scheduling Mechanism This component depends on the *Network Weather Service (NWS)* [11], which is a performance prediction tool that has

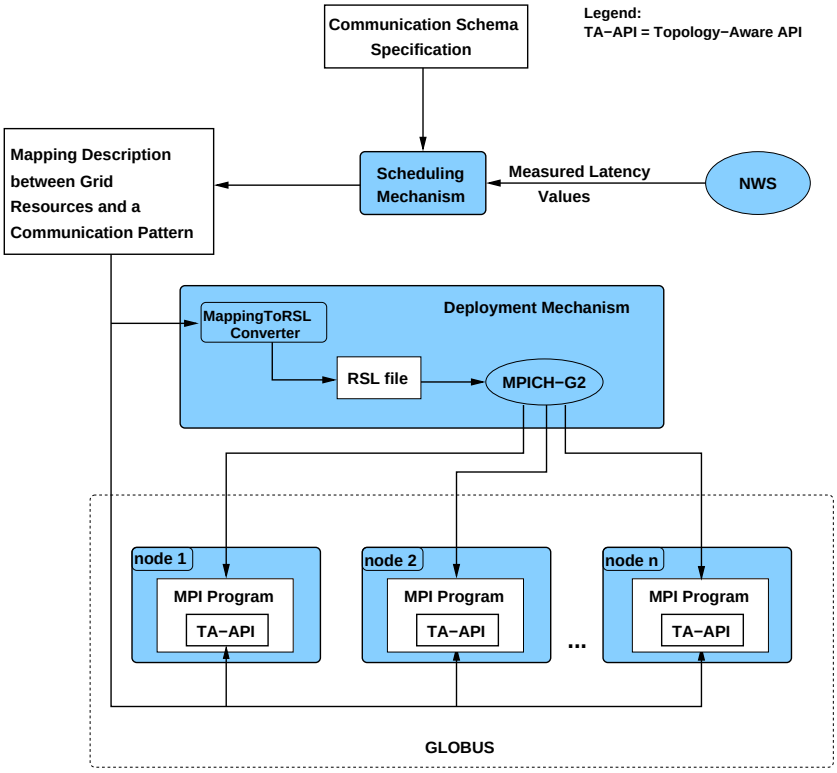

Figure 1: Overview on the Software Framework

become a de facto standard in the grid community. Since the NWS provides all necessary information concerning the utilizable grid resources, the user needs not know any detail of the grid architecture. In addition to these performance characteristics the scheduling algorithm needs a communication schema of a particular application specified 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 (for more details see Section 3). The output of the algorithm is an XML-based *mapping file* which describes a mapping between the grid resources and the given communication pattern.

Deployment Mechanism This mechanism is based on the job starting mechanism of the grid-enabled MPI implementation MPICH-G2 [9]. It expects a mapping file generated by the scheduling mechanism as input which contains among others the name and various locations of

the executable, the designated grid resources and the partition of processes. It then starts in two steps the processes of an application on the grid according to the content of the mapping file:

- First, it distributes via gridFTP the mapping file into the directory `/tmp` on all designated grid machines.
- Then it generates a RSL expression from the mapping file; with the help of this RSL expression, it starts the application on the grid via MPICH-G2.

Topology-Aware API This API is an addition to the MPI interface. Its purpose is to query mapping files and inform parallel programs how their processes are assigned to physical grid resources and which are the designated roles for these processes. It provides information such as in which local group a particular process resides or which are the characteristics of local groups, graphs, trees or rings.

For representing the versatility of our API, we have developed some simple distributed example applications [4, 6] (e.g.: tree-like multilevel parallelism on the grid).

All the three major components of our software framework have completely been implemented and the entire system has already been tested successfully on the sites `altix1.uibk.ac.at` (SGI Altix 350), `lilli.edvz.uni-linz.ac.at` (SGI Altix 4700) and `alex.jku.austriangrid.at` (SGI Altix ICE 8200) of the Austrian Grid.

3 The Scheduling Mechanism

The task of the scheduling mechanism is to find a partition of processes based on the given schema which can be mapped to the available hardware resources such that the assessed use of any slow communication channel is minimized.

This kind of communication-aware mapping is an NP-complete problem which can be only efficiently solved by some kind of heuristic search algorithm. Similar problems have already arisen three decades ago in the mapping of processes to parallel hardware architectures (e.g.: hypercube) [7]. Nowadays the technique of communication-aware mappings is recalled in connection with heterogeneous multi-cluster and grid environments [10]. In this Section we discuss our solution for the problem of the communication-aware mapping in detail whose implementation was completely finished in the last project phase.

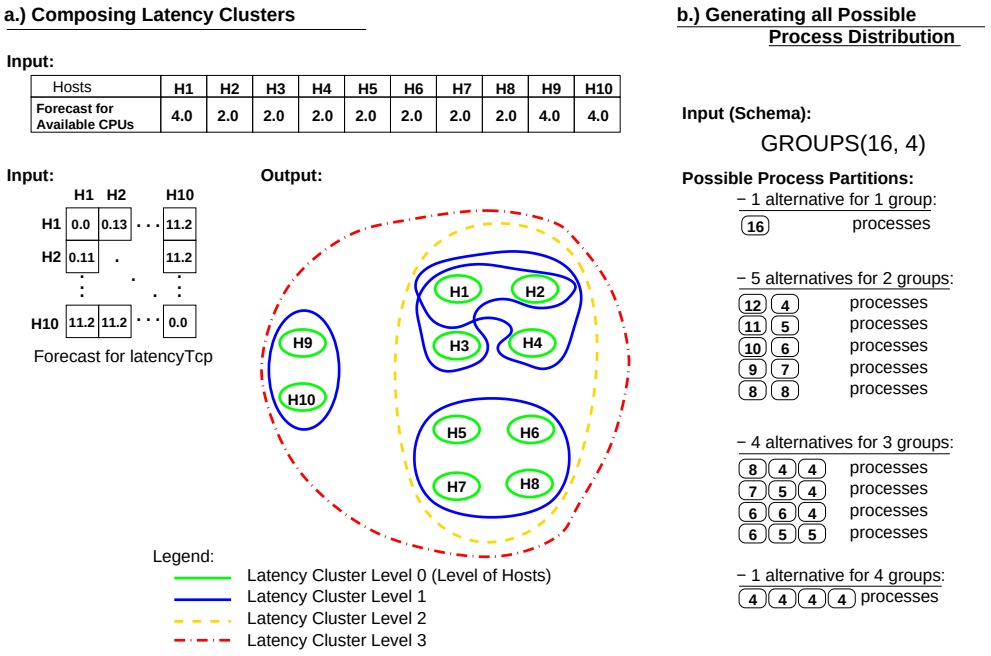

Figure 2: Composing Latency Clusters and Process Partitions

3.1 The Scheduling Algorithm in the Case of the Schema "Groups"

In this section, we describe how the algorithm applied by the scheduling mechanism works in the case of the schema "Groups". The algorithm expects as input 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, and finally a communication schema which specifies the preferred heterogeneous communication patterns of a program. The first three groups of data are provided by the NWS [11] while the schema is given by the user. The algorithm 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 necessarily 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), see Figure 2a. Furthermore each host itself is regarded as a latency cluster with the latency level 0. Each latency cluster has a capacity feature which determines how many processes can be assigned to it at most. This capacity is calculated from the number of CPUs in the latency cluster multiplied with an integer coefficient. The default value of the coefficient is 1, but one can specify a higher value via a command line interface. The generated latency clusters are stored in a list which is sorted according to their maximal latency levels in ascending order (and on the same level according to their capacities in descending 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, see Figure 2b.

4. Finally we map the generated process partitions to some latency clusters according to some compound heuristic (which helps to avoid the combinatorial explosion of possibilities) which roughly works as follows:

   - The process partitions are pre-evaluated. If there exist a latency cluster whose capacity is greater than or equal to the minimal size of groups (given by the schema) and less than or equal to the maximal size of groups (calculated from the schema) then:
     - only those partitions are kept for the mapping which either contains only one group or
     - they have at least one group whose size is equal to the capacity of one of the latency clusters (independently from the latency values the optimal mappings always contain at least one group which fits exactly into a latency cluster).
   - For the mapping the latency clusters are stored in a list in which they are arranged according to their latency level in an ascending order; and on the same latency levels according their capacity in a descending order in the list. A process partition is mapped to some latency clusters group by group (greater groups are assigned earlier). Each group is assigned to a latency cluster whose latency level is minimal and available capacity is large enough for the group. According to some additional low level heuristics a group can be assigned more than one latency cluster if their latency levels are the same (this can result an alternative mapping for a particular partition).

To find a reasonably efficient scheduling for the program in the space of solutions, we associate a cost function to each mapping (between a process partition and some latency clusters). This cost function takes into consideration the following characteristics of the mappings:

- the maximum latency level within the local groups,
- the maximum and the average latency values of all possible links among the local groups.

The algorithm always returns the mapping whose associated cost function is minimal.

3.2 The Scheduling Algorithm in the Case of the Schema "Graph"

In the case of the schema "Graph" the algorithm is slightly different because the number and sizes of local groups are fixed by the given schema. So we count only with the given process partition and we can therefore skip the third step of the algorithm above.

Additionally since the schema "Graph" specifies links among the local groups, in the cost function (in step 4) we apply average and maximum latency values of the pre-defined connections instead of all connections among the groups.

3.3 The Scheduling Algorithm in the Case of the "Tree" and the "Ring" Schemas

Although the number and the sizes of the local groups are not specified in the cases of the "Tree" and the "Ring" schema, but each possible partition contains pre-defined connections among its local groups. Hence, we apply in the cost function (in step 4) the average and maximum latency values of the pre-defined connections instead of all connections among the groups.

Furthermore, in the case of the schema "Tree" we take into account that every local manager process shall be scheduled together with the corresponding leaf group (they are mapped to the same latency cluster).

3.4 Performance of the Algorithm

For testing purposes, we have implemented a virtual topology which is able to substitute for the real measured forecast data of NWS about an available particular grid environment and provides input for the scheduling mechanism.

Nr. of Hosts	Nr. of CPUs	Schema	Execution Time of the Mapping
50	118	Groups{16, 4, 1}	<1s
50	118	Groups{32, 4, 1}	<1s
50	118	Groups{50, 4, 1}	2s
50	118	Groups{64, 4, 1}	85s
50	118	Groups{64, 4, 4}	26s
50	118	Groups{64, 8, 1}	<1s
50	118	Groups{80, 8, 1}	2s
50	118	Groups{80, 8, 2}	1s
50	118	Groups{80, 8, 4}	<1s
100	236	Groups{16, 4, 1}	<1s
100	236	Groups{32, 4, 1}	1s
100	236	Groups{50, 4, 1}	3s
100	236	Groups{64, 4, 1}	1347s
100	236	Groups{64, 4, 4}	351s
100	236	Groups{64, 8, 1}	4s
100	236	Groups{80, 8, 1}	16s
100	236	Groups{80, 8, 2}	9s
100	236	Groups{80, 8, 4}	3s
200	472	Groups{16, 4, 1}	5s
200	472	Groups{32, 4, 1}	18s
200	472	Groups{50, 4, 1}	194s
200	472	Groups{64, 8, 1}	6s
200	472	Groups{80, 8, 1}	31s
200	472	Groups{80, 8, 2}	18s
200	472	Groups{80, 8, 4}	9s

Figure 3: Benchmark Results with Various "Groups" Schemas

In this virtual topology the number of the hosts is scalable, such that we can investigate the efficiency of our scheduling algorithm in the cases of different problem sizes. For the tests we applied virtual topologies consist of 50 hosts (with 118 CPUs), 100 hosts (with 236 CPUs) and 200 hosts (with 472 CPUs) respectively. The scheduler of the TAAG software system were executed on a machine which comprises an Intel Xeon CPU (3.40Ghz).

In the test cases we used various "Graph" and "Groups" schemas. The scheduling of the former ones is always much simpler because in their case only the given fixed process partition has to be mapped to the available grid resources. We have tried numerous different "Graph" schemas the scheduling mechanism always mapped to the three mentioned virtual topologies within 10 seconds, respectively.

Mapping of the "Groups" schemas is a bit more complex, since in these cases the scheduling mechanism must generate all the possible process partitions which satisfies the given schema. Then it has to map all of them to particular grid resources one by one in order to find the best mapping. The benchmark results of the test series concerning the mappings of schema

"Groups is depicted on Figure 3. As it can be seen despite the applied heuristic if the problem size (number of hosts and/or the number of possible process partitions) reaches a certain level the execution time of the scheduler grows exponentially (see the execution time related to schema $Groups\{64, 4, 1\}$). But if we are able to choose the arguments of a schema "Groups" carefully (such that the number of the possible process partition will not be too high), the execution times of the mapping process can be kept only on some couple of seconds (even if the available grid architecture comprises few hundred hosts and CPUs).

4 Comparative Benchmarks with MPICH-G2

To prove the efficiency of the concept of our communication schema based programming and scheduling solution, we performed some comparative benchmarks with TAAG and (pure) MPICH-G2. For this, we had to find a problem that requires a structured communication pattern which can be adapted to heterogeneous network environments, but which can still be implemented easily in pure MPI, too. We chose the well-known n-body problem as a basis of our tests. Here a large number of particles is given (with their positions and masses) as input; the task is to compute the future positions of the particles by taking into account the mutual gravitational attraction among them.

In more detail, we choose as the core of our demonstration an open-source n-body simulation [8]. On the basis of this application we implemented two variants of a toy solution, one in pure MPI and one in TAAG. They solve a special case of n-body problem, where the given particles are arranged into some subsystems called galaxies. The computation of the new positions of all particles of a galaxy is assigned to a disjunct groups of processes. For simplicity, we assume that the galaxies are located far enough from each other such that a collision is not possible between any two of them during the investigated time intervals; the initial setup is prepared accordingly. Due to this assumption a galaxy can be regarded from the other galaxies as a single heavy particle (as long as two galaxies do not approach each other or collide).

So for computing the next position of a particle, on the one hand the actual positions of all other particles of the comprising galaxy and on the other hand the total masses of and the coordinates of the central mass points of the other galaxies are required. The former data is shared among the corresponding processes by applying a ring pipeline communication pattern on the lower level in which $(n-1)*n$ messages are sent in every turn (where

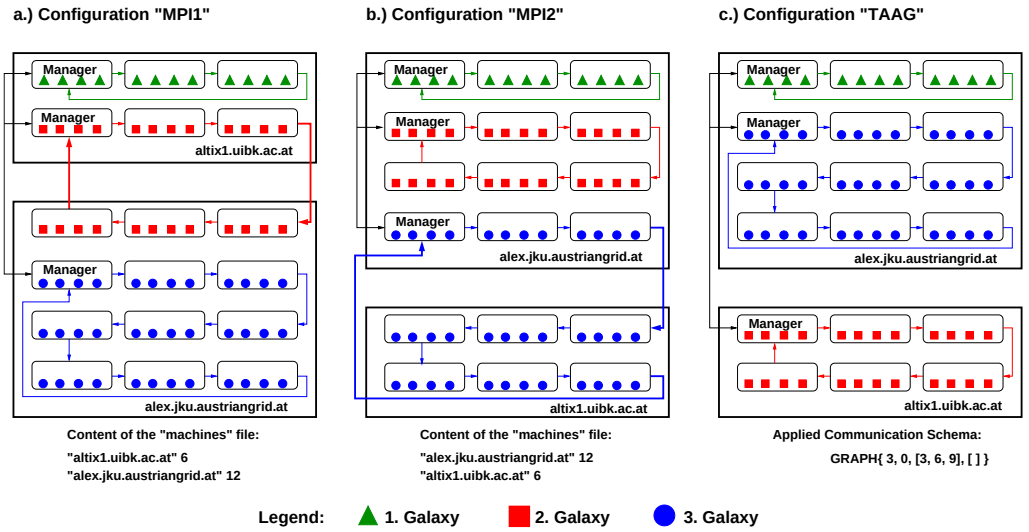

Figure 4: The Applied Test Configurations for the Pure MPI Program and for the TAAG-Based Version.

n is the number of those processes maintaining the particles of a galaxy). The latter data have to be calculated and exchanged among the appointed manager processes of process groups on the higher level (a more general version of the program which will be prepared to take into consideration the collision between two or more galaxies is under development). We have to notice that the implementation of even such a simple two levels algorithm was already much more complicated in pure MPI than with the usage of the API of TAAG.

In the test cases, we used a setup in which the particles are organized into three galaxies such that the sizes of the galaxies are in proportion 1:2:3. Furthermore the particles were always distributed evenly among the processes, so each process maintains the positions of the same amount of particles.

The test cases were executed on two Austrian Grid sites `alex.jku.austriangrid.at` residing in Linz and `altix1.uibk.ac.at` residing in Innsbruck. The former consists of Intel Xeon processors (2.5GHz) the latter comprises Intel Itanium processors (1.4GHz). The two machines are connected via a gigabit WAN connection provided by the Austrian Academic Computer Network (ACONet). In our tests, we used 18 processes and 18 CPUs (12 CPUs on `alex` and 6 CPUs on `altix1`) such that every process can be assigned to a separate CPU (so 3 CPUs are assigned to the smallest galaxy, 6 CPUs to the middle one and 9 CPUs to the largest galaxy).

In MPICH-G2, "machines" files are used to list the computers on which we wish to run our programs. Next to an enumerated machine name in each

Nr. Of Procs.	Particles /Procs.	Time Steps	Configurations			Speedups	
			Mpi1	Mpi2	Taag	Mpi1/Taag	Mpi2/Taag
18	500	10	1.5806s	2.2361s	1.3031s	1.212x	1.715x
18	1000	10	6.0735s	7.7509s	5.1444s	1.18x	1.506x
18	2000	10	23.8034s	29.3047s	20.3422s	1.17x	1.44x
18	4000	10	107.3881s	136.3655s	87.1078s	1.232x	1.565x
18	1000	60	40.4837s	48.6121s	32.7855s	1.234x	1.482x
18	1000	240	161.1072s	203.6982s	133.141s	1.21x	1.529x

Figure 5: Execution Times and Speedups

line, a number appears which specifies the maximum number of processes that can be executed on the machine. The processes are always scheduled to the machine listed first. Then if we intend to use more processes than can be established on the first machine, the remaining processes are scheduled to the subsequent machine in the list. Consequently, the only way to influence the scheduling of the processes from a "machines" is by changing the order of the machine names (it is supposed that the maximum number of executable processes are fixed for each machine).

For scheduling the processes of the MPI program, we applied two different configurations given in "machines" files (see Figure 4).:

- According to the first configuration labeled as *"MPI1"* where the machine `altix1` is listed first, the processes that maintain the particles of the second galaxy are distributed between `altix1` and `alex` (see Figure 4a).

- According to the second configuration labeled as *"MPI2"* where the machine `alex` is listed first, the processes that maintain the particles of the third galaxy are distributed between `alex` and `altix1` (see Figure 4b).

Hence, in both MPI-related configurations from all the $n * (n-1)$ messages sent on the corresponding ring pipeline, $2 * (n-1)$ messages have to be sent via the WAN network connection in every time steps (where n is the number of processes maintaining the particles of a galaxy).

For scheduling the processes of the TAAG-based version of the program we applied the following communication schema

$$GRAPH\{3, 0, [3, 6, 9], []\}$$

which yields the optimal scheduling such that the first and the third galaxies are maintained on the `alex` and the second galaxies are maintained on the

`altix1` (see Figure 4c). Apart from the way how the group of processes are established and assigned to galaxies, the both versions (the pure MPI and the TAAG-based versions) of the program contain the same piece of code (we avoided the usage of any convenient statements or programming structure introduced by our API).

The execution times presented on Figure 5 are average values of 5 computations and do not include the overhead of the job submission in Globus. As it can be seen that the reachable speedup factor is independent how we decreased or increased either the number of the particles or the time interval of the simulations in the test cases. The use of our topology-aware API always speeded up the simulation by a factor of 1.2 compared to the "MPI1" configuration and by a factor of 1.5 compared to the "MPI2" configuration roughly. We have also investigated the case in which the optimal scheduling is possible by MPICH-G2, too (e.g.: on both grid sites 9-9 CPUs are available). In these test cases, the average execution times of the two versions of the application were quite alike. Consequently, we can state that the usage of our TAAG framework offers at least as efficient program execution on the grid as MPICH-G2.

In all likelihood in more realistic test circumstances (in a testbed containing more than two clusters) the achieved speedups would be higher; because in a pure MPI solution the chances that two interacting processes are placed into the same local environment of a grid comprising n pieces of grid nodes (clusters, LANs or single machines) are only $1/n$. While in the TAAG-based solutions most of the communication are done (on the priori identified "often used" communication channels which are always located) within local network environments.

In the future, we intend to develop an efficient distributed n-body simulation (probably on the basis of the hierarchical Barnes-Hut algorithm [3]) whose execution beyond a certain problem size shall require a more complex decomposition of process groups among physical hardware resources. Since in these proposed simulations many to many grid sites interact with each other according to a compound logical hierarchy (instead of the presented simple algorithmic structure consisting of only two machines), we expect a more significant performance gain on the side of the TAAG framework (in comparison with pure MPICH-G2).

5 Conclusions

Summarizing the achievements of the existing topology-aware programming tools (e.g.:MPICH-G2), we can say that they make available the given topol-

ogy 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 communication 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.
- Since our system hides low-level grid-related execution details from the application by providing an abstract execution model, it eliminates some 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.

A drawback of our solution is that the applicable communication patterns cannot be retrieved from the programs. If some schema is not enclosed to a distributed application, its effective scheduling may not be possible at the moment. We propose to overcome this issue in a subsequent version of our software system where the programmer will be forced by the API library to specify a recommended schema (with defined flexibility) via some function calls in the source of the programs. According to our conception, the scheduling mechanism will be able to query this built-in information from the compiled application.

As the next step, we intend to replace the MPICH-G2 in our software framework with its successor called MPIg [2]. By this substitution, our TAAG system will be able to submit and execute parallel programs via the Web-Service architecture of the Globus Toolkit, too (MPIg had only one internal release at the end of 2007, which is freely available for testing and

development purposes). Besides, we also plan to develop on the basis of our TAAG programming framework some grid-distributed parallel applications (e.g.: a distributed n-body simulation based on Barnes-Hut algorithm and some other programs in the fields of the hierarchical distributed genetic algorithms) in cooperation with other research groups.

References

[1] Globus Toolkit. `http://www.globus.org/toolkit/`.

[2] MPIg Release Home. `ftp://ftp.cs.niu.edu/pub/karonis/MPIg/`.

[3] J.E. Barnes and P. Hut. A Hierarchical O(N Log N) Force Calculation Algorithm. *Nature*, 324(4):446–449, December 1986.

[4] 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.

[5] Karoly Bosa and Wolfgang Schreiner. A Supercomputing API for the Grid. In Jens Volkert et al., editor, *Proceedings of 3rd Austrian Grid Symposium 2009*, pages –. Austrian Grid, Austrian Computer Society (OCG), September 28-29 2009. To Appear.

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

[7] Woei-Kae Chen and Edward. F. Gehringer. A Graph-Oriented Mapping Strategy for a Hypercube. In *Proceedings of the third conference on Hypercube concurrent computers and applications*, pages 200–209, New York, NY, USA, 1988. ACM.

[8] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1999.

[9] 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.

[10] Juan Manuel Orduña, Federico Silla, and José Duato. On the Development of a Communication-Aware Task Mapping Technique. *J. Syst. Archit.*, 50(4):207–220, 2004.

[11] 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.