Running parallel applications with topology-aware grid middleware

Pavel Bar[1], Camille Coti[2], Derek Groen[3], Thomas Herault[2], Valentin Kravtsov[1], Assaf Schuster[1], Martin Swain[4]

[1]Technion - Israel Institute of Technology, Technion City, 32000, Haifa, Israel
[2]INRIA Saclay-Île de France, Orsay, F-91893, France
[3]Section Computational Science, University of Amsterdam, Amsterdam, The Netherlands
[4]University of Ulster, Cromore Road, Coleraine, BT52 1SA, Northern Ireland, UK

Abstract

The concept of topology-aware grid applications is derived from parallelized computational models of complex systems that are executed on heterogeneous resources, either because they require specialized hardware for certain calculations, or because their parallelization is flexible enough to exploit such resources. Here we describe two such applications, a multi-body simulation of stellar evolution, and an evolutionary algorithm that is used for reverse-engineering gene regulatory networks. We then describe the topology-aware middleware we have developed to facilitate the "modeling-implementing-executing" cycle of complex systems applications. The developed middleware allows topology-aware simulations to run on geographically distributed clusters with or without firewalls between them. Additionally, we describe advanced coallocation and scheduling techniques that take into account the applications topologies. Results are given based on running the topology-aware applications on the Grid'5000 infrastructure.

1. Introduction

Many real-world systems involve large numbers of highly interconnected heterogeneous elements. Such structures, known as complex systems, typically exhibit non-linear behavior and emergence. Understanding so-called complex systems is increasingly becoming a necessity in science, business and engineering [4].

The term 'complex systems' encompasses a vast variety of biological systems (e.g., gene-regulatory and neural networks), ecosystems (e.g., modeling of lakes), social systems (e.g., simulating crowd behavior), business models (e.g., simulating stock markets), and so on.

Possibly, the most striking common denominator of all complex systems is the fact that the classic experimental methodologies and research approaches plainly fail to allow prediction and control of their behavior. Accordingly, for many real-world systems, *computational modeling and simulation* is therefore the only practical method of developing an understanding of their properties. Such models and simulations often require considerable computational resources.

In this paper we propose a new approach for modeling and executing complex system applications in collaborative grid environments. Just as complex systems consist of many interdependent parts that give rise to non-linear and emergent properties determining their high-level functioning and behavior, so computational models of complex systems may be decomposed into interdependent components, connected according to a specific topology, and capable of exploiting different types of computational resources: this is what we call *topology-aware applications*. The QosCosGrid project has been designed specifically to support the execution of such applications in collaborative grids [7]. We demonstrate the entire process of "modeling-implementing-executing" large scale, topology-aware applications by means of the developed stand-alone components the QosCosGrid resource description schema, QosCosGrid version of Open MPI, and the QosCosGrid Meta-Scheduler.

2 Background: topology-aware applications and middleware

2.1 Topology-aware complex systems applications

Here we describe two parallelized topology-aware computational models of complex systems that we have used for

testing our topology-aware grid middleware.

2.1.1 A topology-aware evolutionary algorithm

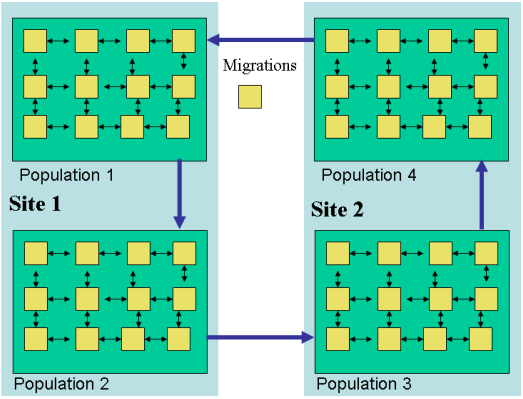

Figure 1. A parallel, topology-aware evolutionary algorithm.

An example of a topology-aware evolutionary algorithm application is depicted in Figure 1. Evolutionary algorithms use an approach inspired by biological evolution to iteratively evolve or optimize a population, where each individual in a population is a potential solution to a technical problem. In this particular example we are interested in discovering a set of parameters that can be used to simulate the dynamic behavior of gene regulatory networks as accurately as possible.

The application shown in Figure 1 has been parallelized in a manner that combines features of both cellular and co-operative-island evolutionary algorithms, thus giving the application a two-layer topology:

1. The island model comprises the higher layer of the evolutionary algorithm. In the island model a genetic population is divided into subpopulations, which are optimized semi-independently from each other – individuals periodically migrate between island subpopulations in order to overcome the problems associated with a single population becoming "stuck" in a local minima and thus failing to find the global minimum.

2. The cellular model is used to optimize the subpopulations that comprise the lower layer of the evolutionary algorithm. In this case individuals in the subpopulations are distributed upon a lattice topology so that each individual exchanges genetic material only with its immediate neighbors.

In this approach more frequent communications take place in the lower layer and less frequent communications in the higher layer. In Figure 1 the evolutionary algorithm comprises four subpopulations distributed over two physical locations or sites. In general, however, the algorithm can be divided into 1 or more subpopulations than can be executed on 1 or more sites, and it is usually the case that 1 subpopulation, with its relatively frequent communication pattern, is restricted to just 1 computational cluster on 1 site. However it is also possible for a population or subpopulation to be executed across 1 or more sites, which may be useful if an optimization problem requires a single large population.

2.1.2 Cosmological N-body

The cosmological N-body code we use in this work is used to perform gravity calculations on large volumes of dark-matter. Inside the code, dark matter is represented as point-particles, which are integrated over time using a combined Tree/Particle-Mesh algorithm [11]. The integrator has been specially optimized to take advantage of the Power6 architecture and the SSE2 instruction set, using the Phantom GRAPE [8]. In addition, it has been successfully ported to run across two supercomputers [9].

The code has been parallelized with QCG-OMPI using a spatial decomposition, with processes containing subvolumes of the universe with equal particle quantities. When run in parallel, the processes exchange particles with neighboring processes to preserve the spatial decomposition of particles throughout the simulation. In addition, some all-to-all communications are performed to perform parallel Fourier transformations, which are required by the Particle-Mesh integrations. As the particles are distributed evenly among the processes, the number of local tree force calculations and particle exchanges are similar for all processes. Consequently, the simulation runs most efficiently on network topologies which are sufficiently balanced in terms of bandwidth, and are able to cope with the large number of messages produced by all-to-all communications.

The simulation we perform in this work contains a total of 256^3 particles mapped to a grid of 128^3 mesh cells. For our experiments, we perform one simulation step and measure the time spent by the simulation.

2.2 Topology-aware grid middleware

In this section we describe the three main middleware components used for facilitating the "modeling-implementation-executing" cycle. The first component is the QosCosGrid resource description model described in section 2.2.1 that is used to model a topology-aware application and its computing and networking requirements. The second component described in section 2.2.2 is the QosCosGrid OMPI (QCG-OMPI) library that is used to implement

the application and in particular its most important part – message passing protocol. The third component described in section 2.2.3 is the QosCosGrid meta-scheduling system, that is used for matching the available grid resources with the application requirements and scheduling the topology-aware jobs on the time axis.

2.2.1 Scheduling Input Format

Large-scale, topology-aware parallel applications may be composed of hundreds to thousands of processes. In practice, these applications can be efficiently described in terms of "process groups" rather than in terms of single processes. This description can significantly minimize the size of a given problem with no loss of description accuracy. For instance, the MPI communicators abstraction usually implies that all the machines in a certain communicator should have similar properties and should be interconnected by similar all-to-all links.

Figure 2 depicts the evolutionary algorithm application presented in section 2.1.1 as described in the proposed framework. In this request, a user asks for three process groups called PG1 – PG3. Each process group is mapped to a computing resource template that describes the properties of the requested identical machines. For example, PG1 requires between 8 and 10 computing resources, each of which conforming to a certain computing resource template. The properties of a computing resource template are described in terms of ranges. For example, the computing resource template requested by both PG1 and PG3 might be specified as follows: [clock rate in range of 2...3GHz], [memory in range of 1...2GB], and [free disk space in range of 2...∞ GB].

Process groups (PGs) are arranged in process communication groups (PCGs) topology, where it is assumed that all processes within a PCG would like to have all-to-all interconnections with certain properties. The quantitative properties of these interconnections are specified by means of the network resource templates. For instance, PCG1 (red circle) contains only processes of Process Group 1 (PG1), which means that all the processes of PG1 must have all-to-all interconnections as described in the PCG1's network resource template. On the other hand, PCG4 (purple circle) contains three process groups – PG1, PG2, and PG3, which means that all the processes of these three PGs must have all-to-all interconnections as described in the PCG4's network resource template. Given any two processes we can determined the quality of their interconnection by looking into the smallest circle (PCG) that contains these two processes. Like the computing resource templates, the network resource templates are described in terms of ranges. A sample template might be specified as follows: [bandwidth in range of 10...54Mbit/sec], and [latency in range of 0.001...0.01sec].

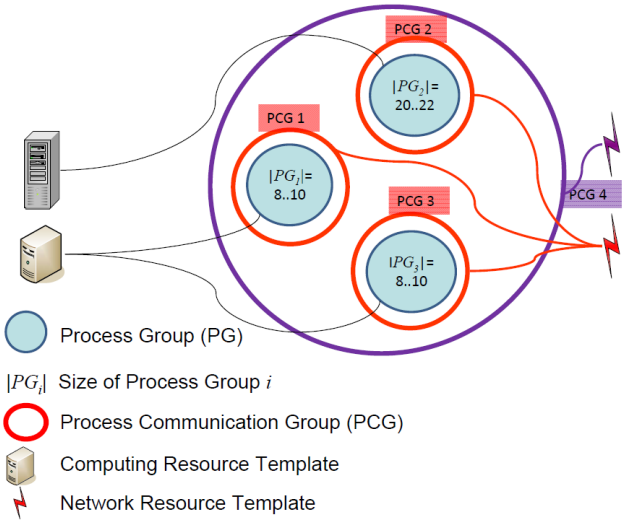

Figure 2. A sample request representation of a topology-aware evolutionary algorithm application

The aforementioned description format is called a resource topology graph (RTG). In addition to the computing and network resources description, the request RTG also contains the estimated runtime of the requested task and optional time bounds for the task's execution.

Like the description of the requests, the offered machines are also described in terms of an RTG. Figure 3 depicts a sample offer which models the three selected Grid'5000 sites. The description of the offered resources contains the details of the computing resources, such as memory and disk space, and the properties of network resources. The network resources are described in terms of resource communication groups (RCGs), which, by definition, means that all the resources within an RCG have identical all-to-all interconnections. An offers RTG also contains the description of resource availability on the time axis.

It is important to stress that the purpose of the request and offer RTGs is to simplify and make more efficient the description of the requested and available resource topologies by users and system administrators. Additionally, to simplify the description and avoid errors, the real values can be replaced by several predefined parameter groups. For instance, the real latency values of the offered resource were replaced by two groups of interest: "LOW/HIGH".

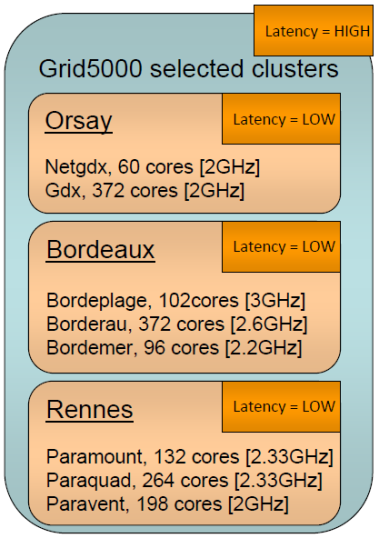

Figure 3. A sample representation of selected Grid'5000 machines

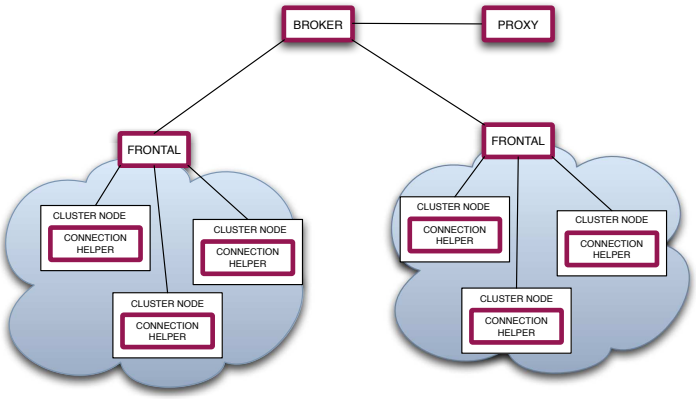

Figure 4. Architecture of the grid infrastructure of QCG-OMPI

process group	PG1	PG1	PG2	PG2	PG3	PG3
rank	0	1	2	3	4	5
depth	2	2	2	2	2	2
colors	PCG4	PCG4	PCG4	PCG4	PCG4	PCG4
	PCG1	PCG1	PCG2	PCG2	PCG3	PCG3

Table 1. Array of colors corresponding to the topology depicted by Figure 2

2.2.2 QCG-OMPI

Parallel applications on grids run into the problem of communicating throughout the grid. Resources must be protected by security equipments such as firewalls and network address translation (NAT) mechanisms, to prevent from external intrusions. On the other hand, processes of an application spanning across multiple administrative domains must be able to communicate with one another.

This issue was addressed in [3]. QCG-OMPI is an MPI implementation targeted to computational grids, based on OpenMPI [5] and relying on a set of grid services that provide advanced connectivity techniques. QCG-OMPI enables communications between administrative domains in spite of firewalls and NATs, without requiring any specific configuration nor privileges. The architecture of the grid infrastructure supporting QCG-OMPI is depicted in Figure 4.

Several interconnection techniques are available: direct connection (when two processes can communicate directly with each other without requiring any specific setting), port range method (when a [known] range of ports is open in the firewall), Traversing TCP (when no port is open in the firewall) and using proxy relays (when none of the aforementioned technique can be used). Traversing TCP has been introduced by PVC [10], a communication middleware that aims at enabling communications across administrative domains.

QCG-OMPI also provides the possibility to retrieve topology information provided by the scheduler. As described in section 3.2, the grid meta-scheduler allocates resources with respect to the requirement provided by the user (section 2.2.1). This topology information is transmitted to QCG-OMPI through an environment variable, and stored by the run-time environment as an MPI attribute.

The application can obtain this information at any moment during the execution (after the MPI library has been initialized and before its finalization), as any MPI attribute. The number of groups a process belongs to is called its *depth*, and is stored under the attribute name `QCG_TOPOLOGY_DEPTH`. The name of the groups defined by the user are called *colors*, and is stored under the attribute name `QCG_TOPOLOGY_COLORS`. QCG-OMPI provides a `QCG_ColorToInt` function to convert alphanumerical colors into integers that can be passed to the `MPI_Comm_split` routine in order to create communicators that fit the topology.

The example of topology depicted by Figure 2 is given by Table 1. In this example each process group contains two processes.

2.2.3 QosCosGrid Meta-Scheduler

The QosCosGrid Meta-Scheduler was build to solve two problems: (1) Coallocation (placement) of the requested processes to the appropriate offered resources; (2) Scheduling of topology-aware applications on the times axis. In [6] it was shown that even the coallocation of a single multi-

processors, topology-aware job to homogeneous clusters in a single time slot is NP-complete and cannot be approximated by polynomial time algorithms. It is also clear that scheduling of such topology-aware jobs is NP-complete as it involves coallocating jobs according to a specified order.

The detailed implementation of the scheduling and coallocation algorithms are presented elsewhere, while here we will only describe the formalized model of the scheduling and coallocation problems.

The scheduling system has three inputs: resource topology graph (RTG) of requests, RTG of offers, and the scheduling time slot. The scheduling time slot specifies the time bounds of the scheduling round. It is dictated by the scheduler invoker and may vary according to whether the scheduling scenario is, for instance, recurrent or ad hoc.

2.2.4 Formalizing the resource request

The RTG of a requested task contains the description of required resources, the task's estimated $runtime$, and optional runtime bounds: $requested_{start}$ and $requested_{end}$ between which the task must be executed ($requested_{end} - requested_{start} \geq runtime$). This RTG is translated into a graph $G = (V, E)$, where $|V| = n$ vertexes denote the task's process groups (PGs). Each PG (v_i) requires a number of identical resources (a cluster of identical machines) according to its size. The sizes of PGs are given as ranges denoted by the vectors $CAP_{min} = [cap_{min_1}, \ldots, cap_{min_n}]$ and $CAP_{max} = [cap_{max_1}, \ldots, cap_{max_n}]$. The properties of each machine in the requested clusters are presented as property vectors $C_{min} = [c_{min_1}, \ldots, c_{min_n}]$ and $C_{max} = [c_{max_1}, \ldots, c_{max_n}]$, where c_{min_i}, c_{max_i} denote the minimal and the maximal quantitative properties of each required computing resource in cluster v_i (e.g., FLOPS). Different quantitative properties might be described by multiple property vectors.

The process communication groups are translated into graph edges E, denoted by n-by-n adjacency matrices B_{min} and B_{max}, where $b_{min_{ij}}$, and $b_{max_{ij}}$ refer to the minimal and maximal connectivity level of the machines in cluster v_i and v_j. Matrices B_{min} and B_{max} represent the communication latency between and within the requested clusters as requested by the user.

2.2.5 Formalizing the resource offer

Analogously, an offer RTG is translated to a graph $\hat{G} = (\hat{V}, \hat{E})$, where $|\hat{V}| = m$. Each vertex $\hat{v_j}$ represents a cluster of identical offered machines. A capacity vector $\hat{CAP} = [\hat{cap}_1, \ldots, \hat{cap}_m]$ denotes the number of available machines in each cluster. To simplify the formalization, we will assume that each machine has a single CPU, although extending the model to handle an unrestricted number of CPUs per machine is straightforward, moreover, in the following sections, this assumption will be discarded. The quantitative properties (e.g., FLOPS) of machines in each cluster are denoted by the vector $\hat{C} = [\hat{c_1}, \ldots, \hat{c_m}]$. In contrast to the request RTG, the offer RTG has no ranges, as it represents the values of real machines in the grid.

An m-by-m adjacency matrix $\hat{B}$ represents the edges' properties (e.g., the latency within and between the m clusters in the grid), assuming identical connectivity properties between all the machines in each cluster. Additionally, each machine r in cluster j has a list of time slots $T = \{(t^1_{start_{rj}}, t^1_{end_{rj}}), (t^2_{start_{rj}}, t^2_{end_{rj}}), ...\}$ during which the resource is marked as "available."

2.2.6 Formalizing the output

The goal of the grid-level scheduler is to provide a scheduling plan for a large number of requests and offers. For each task, we define an n-by-m allocation matrix X in which the term $X_{ij} = k$ represents an allocation of k processes of a requested process group v_i to an offered cluster $\hat{v_j}$. The allocation matrix X must satisfy the following constraints:

$$\forall i : 1 \leq i \leq n, \quad cap_{min_i} \leq \sum_{j=1}^{m} X_{ij} \leq cap_{max_i}, \quad (1)$$

denoting that each process group must be served by a range of $cap_{min_i} \ldots cap_{max_i}$ offered resources;

$$\forall j : 1 \leq j \leq m, \quad \sum_{i=1}^{n} X_{ij} \leq \hat{cap}_j, \quad (2)$$

denoting that an offered cluster j can serve at most $\hat{cap}_j$ processes;

$$\begin{array}{l} \forall i, j : 1 \leq i \leq n, 1 \leq j \leq m, \\ \quad \hat{c_j} > c_{max_i} \vee \hat{c_j} < c_{min_i} \Rightarrow X_{ij} = 0 \end{array} \quad (3)$$

denoting that if offered resources in cluster j do not match the requested process group i, then no allocations are allowed;

$$\begin{array}{l} \forall i, j, k, l : 1 \leq j, l \leq m, 1 \leq i, k \leq n, \\ \quad sgn(X_{ij}X_{kl}) b_{min_{ik}} \leq sgn(X_{ij}X_{kl}) \hat{b_{jl}} \leq b_{max_{ik}}, \end{array} \quad (4)$$

denoting that the connectivity (edge) properties of the requests must match the connectivity properties of offers[1] (formalized only for the latency metric due to space limitations); and

$$\begin{array}{l} \forall i, j : 1 \leq i \leq n, 1 \leq j \leq m, \\ \quad X_{ij} \in \{0, \ldots, \min(cap_{max_i}, \hat{cap}_j)\}, \end{array} \quad (5)$$

[1] $sgn(X_{ij}) = \begin{cases} 1 & \text{if } X_{ij} > 0 \\ 0 & \text{if } X_{ij} = 0 \end{cases}$, as $X_{ij} \geq 0$

denoting that the number of allocations is an integer.

In addition to the constraints on matrix X, there are the following time constraints:

$$task_{start} \geq requested_{start} \wedge task_{end} \leq requested_{end} \wedge task_{end} - task_{start} = runtime, \quad (6)$$

denoting that the actual task's start and end times must fit the given bounds; and

$$\forall i,j : 1 \leq i \leq n, 1 \leq j \leq m, \quad X_{ij} > 0 \Rightarrow \exists R_j = \{r_{1j}, r_{2j}, \ldots\}, \forall r \in R_j, \exists k : task_{start} \geq t^k_{start_{rj}} \wedge task_{end} \leq t^k_{end_{rj}} \wedge |R| = X_{ij} \quad (7)$$

denoting that only "available" resources can be matched.

The output of the QosCosGrid Meta-Scheduler is the scheduling plan where each given job i is scheduling to run at time t_i (if any) on a set of resources R_i. It is possible to measure the quality (utility) of a given scheduling plan by many different parameters such as: resource utilization percentage, fairness degree, task satisfaction percentage (where task is assumed to be more "satisfied" if it got a higher percentage of requested resources), jobs makespan (end time of the last scheduled job), etc. QosCosGrid Meta-Scheduler can support any aforementioned utility function as well as any combination of two or more utility functions.

3 Experimental Methodology

3.1 Implementing the applications with QCG-OMPI

Processes of the application can obtain at run-time the topology description as represented by table 1 by obtaining the appropriate MPI attributes. They can decide what they have to do regarding their color at a given depth, and adapt their computation pattern. They can also use colors to adapt their communication patterns. In particular, they can build communicators that fit the requested communication groups using the MPI routine `MPI_Comm_split()`. The color used by `MPI_Comm_split()` is the integer translation of the QCG alpha-numeric color, given by the `QCG_ColorToInt()` routine. Processes that do not belong to any communication group at a given depth are assigned the MPI_UNDEFINED color, and create an invalid communicator, as specified by the MPI standard.

Communications within process groups ought to use these topology-aware communicators, as presented in [2].

3.2 Coallocation of Topology-Aware Applications

The algorithm of the coallocation procedure used for mapping the aforementioned applications to the available grid resources is described in [6]. Although this allocation algorithm is heuristic in its nature, in all the tests performed on the Grid'5000 infrastructure it provided the optimal results for coallocation of a single job because the available resources are in fact clustered into relatively small number of homogeneous clusters.

During all the experiments, the offered machines included 1596 cores arranged in 8 clusters of homogeneous machines spread on 3 physical sites (Orsay, Rennes, and Bordeaux). One of our assumptions in the allocation procedure of the evolutionary algorithm described in section 2.1.1 was that within the clusters we can expect relatively "good" connectivity level, while between the clusters this connectivity is relatively "bad". This assumption is based on the fact that other users are also utilizing inter-clusters links thus reducing the available bandwidth. Thus, for example, the hard constraint of the biological evolutionary algorithm was that an island must not be divided between clusters, while different islands can be allocation to the same or different clusters on different sites.

Another constraint posed by the evolutionary algorithm application was that the provided resources per island should be as homogeneous as possible in terms of the CPU clock rate, while giving the preference to strongest CPUs.

4 Experimental Results

Experiments were run on the academic grid Grid'5000 [1], a nation-wide experimental grid dedicated to research. We used three sites, located in Orsay, Rennes and Bordeaux, distant by several hundreds of kilometers from each other. In Orsay the GdX cluster was used, using AMD Opteron CPUs running at 2.4 GHz, in Bordeaux the Bordereau cluster was used, using dual-core AMD Opteron CPUs running at 2.6 GHz, and in Rennes the Paradent cluster was used, using Intel Xeon CPUs running at 2.5 GHz. Nodes are interconnected within each cluster by a Gigabit Ethernet switch, and clusters are interconnected by the Renater French Education and Reseach Network using 10 Gb/s dark fiber.

4.1 Evolutionary Algorithm

Figure 5 displays average execution times for the evolutionary algorithm (shown in Figure 1) to execute 300 iterations using either 1 or 2 clusters located in Orsay and Bordeaux. As the number of cores is increased, so the population size increases because each individual runs on a seperate core – and the advantage of a larger population is that optimal solutions are discovered more quickly. The clusters run at different speeds (2.4 Ghz and 2.2 Ghz respectively).

The execution times are shown in Figure 5, first when running single populations on each cluster independently

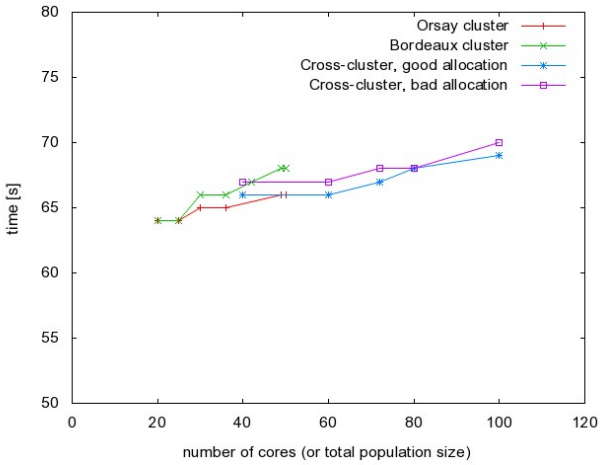

Figure 5. Evolutionary algorithm scale-up.

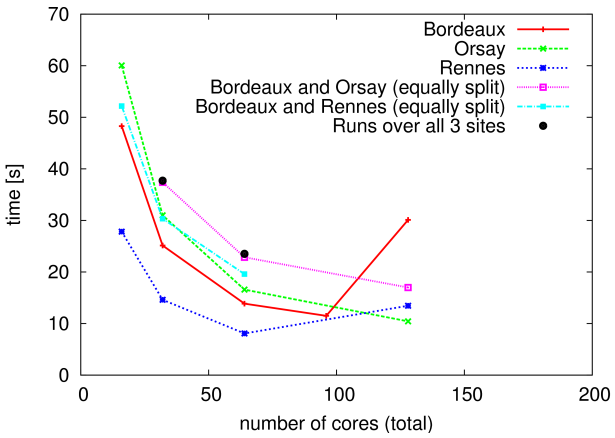

Figure 6. Cosmological N-body speed-up.

from each other, and then when performing a cross-cluster run with the populations split into 2 subpopulations.

In the "good allocation" runs we have placed each subpopulation entirely on one of the clusters according to the output of the coallocation algorithm, so that the frequent communications take place within a single cluster; while in the "bad allocation" runs we have placed each subpopulation on both clusters, so that the communications must travel between Orsay and Bordeaux at each iteration. The figure shows a fairly constant increase in time between these two runs due to this communication overhead. An important feature of the figure is that it shows that the scalability of a properly coallocated application (in a way that takes into account the application topology) is almost equal to the scalability of an application executed on a single cluster.

4.2 Cosmological N-body Integration

We have performed a series of runs with the Cosmological N-body code to measure the performance when it is mapped across multiple sites and using different resource topologies. For these experiments we have used the three aforementioned clusters, using between 16 and 128 processes in total for each run. The performance results of these runs are shown in Figure 6. Here the wall-clock time is given as a function of the number of cores. Results are shown for different topologies, with the thick black points indicating two runs over all three sites. The 32 core run was performed using 8 cores in Bordeaux, 16 cores in Orsay and 8 cores in Rennes, whereas the 64 core run was performed using 24, 32 and 8 cores respectively.

Due to the low-level optimizations implemented in the code, the performance is strongly dependent on the underlying hardware. The code performs best on the Rennes site for a low number of cores, but as the number of cores is scaled up, the performance in Orsay becomes comparatively better. Consequently, a single-site simulation using 128 cores runs most efficiently in Orsay. During our runs in the Bordeaux site we were unable to use the most optimal network. This is reflected in our results by a measured inferior scalability over the number of cores. However, on a small number of cores, where communication is less dominant, the code performance is at the mid-point between the results obtained at Rennes and Orsay. When the code is run across multiple sites, it is about 20% slower when compared to a run with the same number of cores on the slowest participating site. However, a notable exception to this trend can be observed in the performance of the runs using 128 cores. Here we see that the run across Bordeaux and Orsay with 64 cores per site performs better than the single site run using 128 cores in Bordeaux. The run across sites performs better in this case because fewer processes are communicating over the local network in Bordeaux, thereby limiting the network load and preventing congestion from occuring.

5 Discussion

Resources in a grid differ in terms of computing properties, such as CPU speed, available memory, and how these resources are aggregated into topologies. To have good performances on grids, applications must be written in a way that they are adapted to the topology of the grid.

For example, consider an evolutionary algorithm that we want to study using a certain number of islands, and using different numbers of islands. The problem is twofold: (1) How to obtain the best performance using a given number of islands; (2) How to schedule the successive experiments in order to optimize the total makespan for the series of experiments.

The first part of the problem consists in coallocating the application processes to the proper computing resources in a way that will also respect the connectivity requirements of an application. For instance, if we want to run an evolutionary algorithm on two islands, processes must be scheduled on a set of resources that are not more partitioned than two resource groups. Typically, each process group must be scheduled on a cluster.

The QosCosGrid approach allows the user to give enough information to the grid meta-scheduler in order to perform topology-aware coallocation. Other approaches like MPICH-G2 or PACX-MPI discover the topology at run-time, and do not guaranty any property on the resource topology. For instance, an application can end up being scheduled on five groups of resources, discover it at run-time, build one island per group and converge differently than if it had been using two islands. Another way to program it would be to use a fixed island size, for example splitting the processes into two groups of equal number of elements, but then processes can be located anywhere.

The QosCosGrid middleware makes sure the application is deployed on resources along an appropriate topology, and provides the application with the possibility to retrieve at run-time the topology that has been asked for. This approach makes sure the aforementioned application is scheduled over a number of resource groups that match the requested number of process groups (in this example, the number of islands) and let processes know at run-time which group they belong to.

The second problem concerns the optimization of the scheduling plan in order to minimize the total makespan or any other utility function. If an application has to be executed several times using various numbers of islands, and the available resources span on clusters of various sizes, the grid meta-scheduler can compute a fair makespan that respects the requirements of the application and is usually reasonably close to the optimal solution.

6 Conclusion

In this paper, we have presented a full software stack for execution of complex system simulations on computational grids. This solution consists of a resource description schema for application modeling, an implementation of message passing protocol that supports cross-cluster executions, and a scheduling framework for coallocation and scheduling of topology-aware applications.

We presented two sample applications enabled for this system whose performance results show that a properly modeled, large-scale, tightly-coupled application can benefit from the computational and storage power of grids.

The biggest advantage of this work is that the effort needed to match the application's communication and computing scheme with the underlying topology is pushed to the meta-scheduler, which allows the programmer to use static load-balancing reducing the complexity of the application. This is opposed to the previous attempts to adapt the applications to the physical topology requiring a dynamic adaptation of the application to the actual physical topology of the infrastructure.

References

[1] F. Cappello et al. Grid'5000: A large scale and highly reconfigurable grid experimental testbed. In *Proc. of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 99–106, Seattle, Washington, USA, Nov. 2005. IEEE/ACM.

[2] C. Coti, T. Herault, and F. Cappello. MPI applications on grids: a topology-aware approach. In L. N. in Computer Science, editor, *Proceedings of the 15th European Conference on Parallel and Distributed Computing*, volume 5704, pages 466–477, Delft, the Netherlands, August 2009.

[3] C. Coti, T. Herault, S. Peyronnet, A. Rezmerita, and F. Cappello. Grid services for MPI. In *8th International Symposium on Cluster Computing and the Grid (CCGRID'08)*, pages 417–424. IEEE Computer Society, 2008.

[4] R. L. Flood and E. R. Carson. *Dealing with complexity: an introduction to the theory and applications of systems science*. Plenum Press, New York, NY, USA, 1993.

[5] E. Gabriel et al. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[6] V. Kravtsov, M. Swain, U. Dubin, W. Dubitzky, and A. Schuster. A fast and efficient algorithm for topology-aware coallocation. In *ICCS*, pages 274–283, 2008.

[7] K. Kurowski et al. Complex system simulations with qoscosgrid. In *ICCS*, pages 387–396, LA, USA, 2009.

[8] K. Nitadori, J. Makino, and P. Hut. Performance tuning of N-body codes on modern microprocessors: I. Direct integration with a hermite scheme on x86_64 architecture. *New Astronomy*, 12:169–181, Dec. 2006.

[9] S. Portegies Zwart, T. Ishiyama, D. Groen, K. Nitadori, J. Makino, C. de Laat, S. McMillan, K. Hiraki, S. Harfst, and P. Grosso. Simulating the universe on an intercontinental grid of supercomputers. *IEEE Computer (submitted)*, 2009.

[10] A. Rezmerita, T. Morlier, V. Néri, and F. Cappello. Private virtual cluster: Infrastructure and protocol for instant grids. In *12th International Euro-Par Conference*, volume 4128 of *Lecture Notes in Computer Science*, pages 393–404, Dresden, Germany, 2006. Springer.

[11] K. Yoshikawa and T. Fukushige. PPPM and TreePM Methods on GRAPE Systems for Cosmological N-Body Simulations. *PASJ*, 57:849–860, Dec. 2005.