Towards a scheduling policy for hybrid methods on computational Grids

Pierre Manneback[1], Guy Bergère[2], Nahid Emad[3], Ralf Gruber[4], Vincent Keller[4], Pierre Kuonen[5], Tuan Anh Nguyen[5], Sébastien Noël[1], and Serge Petiton[2]

[1] Faculté Polytechnique de Mons and CETIC, Mons, Belgium
{Pierre.Manneback,Sebastien.Noel}@fpms.ac.be
[2] INRIA-Futurs, LIFL, USTL, Villeneuve d'Ascq, France
{Bergere,Petiton}@lifl.fr
[3] Laboratoire PRISM, UVSQ, Versailles, France
nahid.emad@prism.uvsq.fr
[4] Département STI-SGM, EPFL, Lausanne, Switzerland
{Ralf.Gruber,Vincent.Keller}@epfl.ch
[5] University of Applied Sciences of Fribourg, Fribourg, Switzerland
{Tuan.Nguyen,Pierre.Kuonen}@eif.ch

Abstract. In this paper, we propose a cost model for running particular component based applications on a computational Grid. This cost is evaluated by a metascheduler and negotiated with the user by a broker. A specific set of applications is considered: hybrid methods, where components have to be launched simultaneously. [6]

1 Introduction

Hybrid methods mix together several different iterative methods or several copies of the same method in order to solve efficiently some numerical problems. They can be considered as alternative to classical methods if two properties are matched: the convergence of the hybrid method has to be faster than each individual method and merging cost between methods has to be low in comparison with the convergence speed-up.

Hybrid methods are used in different fields such as combinatorial optimization [7], numerical linear algebra [5, 3] and general asynchronous iterative schemes [1]. They are well suited for large parallel heterogeneous environments such as Grids, since every method can run asynchronously at each own pace. In order to accelerate convergence, they need however regular interactions, and therefore a suitable coschedule has to be proposed. In this paper, we introduce the coscheduling problem for a specific class of hybrid methods. We start in the next

[6] This research work is carried out under the FP6 Network Of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265). It is a collaborative work between several partners of Resource Management and Scheduling Virtual Institute (WP6).

section by describing a proposal for a cost model. We pursue in section 3 by describing a class of hybrid methods (hybrid iterative methods for linear algebra) as a case study for the scheduling. We continue in section 4 by presenting POP-C++, which is a programming environment easying the development of parallel applications on the Grid, and is well suited to deploy hybrid methods.

2 Description of a cost model

Computational grids offer a considerable set of resources to run HPC applications. Resource management and scheduling are of paramount importance to exploit economically these grids. We have to avoid for instance to run non adapted applications on some resources and spoiled them.

Let consider one parallel application A composed of C_1, C_2, ..., C_n components (i.e. parallel tasks), which interact together (inter-parallelism). Components have an internal parallel structure (intra-parallelism) and can be composed and described by a workflow [2]. A computational grid is composed of R_1,...,R_r resources, each of them disposing of a local resource information system.

We assume that one component C_k can only be placed on a certain amount of nodes on one or more resources R_i (each composed of p_i nodes). Each resource can run one or more components. A node is composed of one or a few processors. We denote P_{ij} the node j on a resource i. At any time, each node can only be devoted to at most one component. We will suppose also that the multiprocessor resources have a distributed-memory architecture. The Grid architecture we focus on is a dedicated computational Grid, composed of several clusters.

A schedule S will be denoted by a list of mappings

$$C_k \rightarrow (\{P_{ij}\}_k, t_k^{start}, t_k^{end}) \qquad\qquad k = 1, \ldots, n \tag{1}$$

t_k^{start} and t_k^{end} are respectively the starting time and the estimated ending time of a component C_k on the set of nodes $\{P_{ij}\}_k$.

The workflow is defined by two types of constraints:

- a partial order precedence relation $\prec$, $C_{k_1} \prec C_{k_2}$ meaning that $t_{k_1}^{end} < t_{k_2}^{start}$. We denote by $\mathcal{P}$ the set of all couples (k_1, k_2) such that $C_{k_1} \prec C_{k_2}$. These constraints have to be strictly respected.
- a simultaneity relation $\simeq$, $C_{k_1} \simeq C_{k_2}$ meaning that $t_{k_1}^{start}$ and $t_{k_2}^{start}$ should be equal. We denote by $\mathcal{S}$ the set of all couples (k_1, k_2) such that $C_{k_1} \simeq C_{k_2}$.

This last set of constraints is very important for hybrid methods where different collaborative components should be launched at the same time.

The basic model for scheduling the components C_1, C_2,..., C_n on the grid for one HPC application A is defined as:

Find a schedule S such that it minimizes cost(A,S) with respect to constraints:

$$\forall P_{ij} \in \{P_{ij}\}_k, P_{ij} \; is \; admissible \; for \; running \; C_k \tag{2}$$

$$t^{end}(A,S) \leq t^{end}_{max}(A) \tag{3}$$

$$cost(A,S) \leq cost_{max}(A) \tag{4}$$

$$C_{k_1} \prec C_{k_2} \qquad \forall (k_1,k_2) \in \mathcal{P} \tag{5}$$

$$C_{k_1} \simeq C_{k_2} \qquad \forall (k_1,k_2) \in \mathcal{S} \tag{6}$$

The function $cost(A,S)$, which represents the cost for the user, has to take account of different parameters: cpu time, elapsed time, communication volume, storage cost, number of used processors on a resource R_i, usage cost of this resource, execution time interval, power consumption, etc. It is evaluated by the metascheduler, on the basis of the information provided by the local schedulers.

It will be the task of a resource broker to propose a suitable allocation and schedule. This broker will invoke a metascheduler, which will call the local resource information system on each resource or pool of resources. Each local scheduler will reply by a service message describing availabilities, nodes specifities (e.g. softwares and libraries) and reservation costs (cost per hour for each type of nodes, cost for a certain volume of transferred data, cost for power consumption, etc.). The metascheduler, by the mean of the data repository, will be able to select suitable schedules that will meet users and resource administrators requirements. We intend to exploit the UniCORE/MetaScheduler/ISS Grid middleware [6]. While this approach is feasible for small sets of resources, it would not scale up to large scale Grids, where a discovery and preselection phase would have to be implemented.

The admissibility of the allocation of A (2) lies in that all nodes in $\{P_{ij}\}_k$ have to meet all the requirements of C_k in terms of permissions, operating system, software, licenses, storage, memory, minimal and maximal number of processors and local policy.

The end user will give his requirements by specifying two parameters: the maximal cost $cost_{max}(A)$ that he wants to pay for running his application and the deadline upper limit $t^{end}_{max}(A)$. The metascheduler will propose suitable resources for each component C_k, with table of costs, starting time and ending time. If both user requirements $cost_{max}(A)$ and $t^{end}_{max}(A)$ can not be simultaneously met, schedule bids will be proposed in two groups: the first one with schedules respecting the cost limit; the latter one with schedules respecting the ending time limit. We will not consider here the problem of rescheduling components or preemption of resources.

In order to illustrate the cost model, let us consider an application with two components C_1 and C_2 in a serial workflow and a grid made of 3 resources R_1, R_2, R_3 composed of 16, 4, and 16 computing nodes, respectively. The collected information about resources (number of processors available during a certain time interval, available librairies and cost) is presented in Table 1. In this example, the resource R_3 is the most expensive one. The cost is defined by each resource administrator and the high cost of a resource will generally means a high performance network and high performance nodes. An administrator can

resource	#proc	t^{start}	t^{end}	supplied libraries	cost / (t.u. × proc)
R_1	12	1	20	L_2	20
R_2	4	1	6	L_1, L_2, L_3	20
R_2	4	7	20	L_1, L_2, L_3	15
R_3	8	5	15	L_2, L_3	25

Table 1. Collected information from each local scheduler by the metascheduler

impose high cost without proposing high performance resources to keep the resource unused (e.g. for local usage). R_3 is assumed to be perfectly scalable and its per processor computing time is 25% lower than R_2. On resource R_2, scalability is linear until 2 processors, and has a value of 3.2 on 4 processors. Each local scheduler can impose varying costs depending on specified time intervals. For instance, the R_2 administrator encourages the use of R_2 after time 6 by applying attractive costs. User requirements are identified by cost and completion time bounds. In this example, user has fixed $cost_{max}(A)$ at 540 units and $t^{end}_{max}(A)$ at 10 time units. The user do not give any information concerning the number of required processors : this kind of information will be provided by the Gamma model of the ISS [4]. We consider that C_1 needs the library L_1 and C_2 needs libraries L_2 and L_3. Therefore, C_1 is admissible on resource R_2 only and C_2 on resources R_2 and R_3. We suppose that the processor time for C_1 on R_2 is 8 time units, independent of the number of processors used, and the processing time of C_2 is 16 units on R_3, thus 20 units on a 2 processor R_2, and 25 units for a 4 processor R_2. Such information can be obtained through the Gamma model. The resource broker will gather from the metascheduler and the local schedulers potential schedules of the type illustrated at Table 2.

#	comp	resource	#proc	#start	#end	cost
1	C_1	R_2	2	1	4	160
2	C_1	R_2	2	5	8	140
3	C_1	R_2	2	7	11	120
4	C_1	R_2	4	7	8	120
5	C_1	R_2	4	1	2	160
6	C_2	R_2	2	5	14	320
7	C_2	R_2	2	9	18	300
8	C_2	R_2	4	3	$8\frac{1}{4}$	455
9	C_2	R_2	4	7	$12\frac{1}{4}$	375
10	C_2	R_3	8	5	6	400
11	C_2	R_3	8	9	10	400

Table 2. Scheduling of components on available and admissible resources.

Taking into account the precedence constraint ($C_1 \prec C_2$), some bids can be proposed for which:

1. $cost_{max}(A)$ is respected
2. $t^{end}_{max}(A)$ is respected
3. Both criterions are respected

Therefore, the metascheduler will propose three bids as shown in Table 3, all of them respecting the sequential workflow. The first one is of minimal cost of 420 cost units, but lasts 18 time units instead of 10, as requested by the user. The second one has a minimal ending time of $t^{end} = 6$, but costs 560 units instead of 540 demanded by the user. The last one respects both constraints.

#	sched	t^{end}	cost
$bid1$	4 → 7	18	420
$bid2$	5 → 10	6	560
$bid3$	4 → 11	10	520

Table 3. Scheduling bids proposed by the broker. The notation i → j means that scheduling is based on rows i and j of Table 2.

Our proposed allocation and scheduling problem is combinatorial. Some heuristics will have to be exploited in order to explore the set of admissible schedules and propose consistent bids. The idea is to develop a *Contract Manager* between the user and the Grid. Different bids can be proposed to the user, with different costs respecting the user requirements. The plausibility of the given ending time should be estimated in such a way that realistic contract offers can be proposed. Therefore, an evaluation phase can be required in order to evaluate the size of the user application A (computation and communication requirements). The usage of a data repository as proposed in the Intelligent Scheduling System [6] will be necessary for this phase.

One major difficulty is the necessary coallocation and coscheduling of communicating components. Here we will consider a particular class of hybrid iterative methods. This will serve us as a case study and will be presented in the next section.

3 Case study

3.1 Hybrid iterative methods for linear algebra

Hybrid methods combine several different numerical methods or several copies of the same method parameterized differently to solve efficiently some numerical scientific problems. For example, both convergence acceleration techniques and preconditioning methods could be used to develop a hybrid method using the first way. An asynchronous parallel hybrid method has some properties such as asynchronous communications between its coarse grain subtasks, fault tolerance and dynamic load balancing which make this kind of methods well-adapted to

the Grid computational environments. The asynchronous hybrid algorithms can be easily implemented on a cluster of heterogeneous machines or on a Grid as it exhibits a coarse grain parallelism. These machines can be sequential, vector, or parallel. The number of iterations to convergence of the main process of a hybrid method can be reduced by combining results from other processes at runtime.

Each collaborative copy of a method, taking part in such hybrid computation is called a co-method and can be represented by a component. The natural parallelism of these components constituting a hybrid method can be different. An example of the second kind of the hybrid methods to compute a few eigenpairs of a large sparse non-hermitian matrix is the multiple explicitly restarted Arnoldi method (Multiple ERAM or MERAM) [3]. This method is based on a multiple projection of the explicitly restarted Arnoldi method (ERAM). Every collaborative component representing a co-method and taking part in such hybrid computation projects the initial problem in a different subspace. Each co-method calculates an approximated solution of the problem on its own subspace. The collaborative process is activated at the end of each iteration of every co-method. At this stage, the available intermediary results of the other co-methods are also considered in order to determine a better projection subspace for the next iteration. This process is depicted in Figure 1 in which $HR(1_{k_1}, 2_{k_2}, \ldots, \ell_{k_\ell}) = HR(U_{k_1}^{m_1}, \ldots, U_{k_\ell}^{m_\ell})$ denotes the hybrid restarting strategy taking into account $U_{k_i}^{m_i}$. Where $U_{k_i}^{m_i}$ is the set of the intermediary eigenvectors computed by the k_ith restart of the ith co-method (for $i = 1, \ldots, \ell$ and $k_i = 1, 2, \ldots$). In this figure, we suppose that we have to compute an approximation (λ^m, u^m) for the eigenpair (λ, u) of the matrix A. Thus, $U_{k_i}^{m_i}$ represents just the approximated eigenvector $u_{k_i}^{m_i}$ computed by the k_ith restart of the ith co-method.

Many algorithms based on the Krylov subspace methods, like ERAM, GMRES, Generalized Conjugate Residual method,... can be executed concurrently as a hybrid method. Indeed, once a co-method ends an iteration, the just computed information can be sent to the others to be incorporated in their next restarting strategy. Thus, each co-method can benefit from two types of results: its `own` results and the `remote` ones, issued from the other co-methods in collaborating computations.

3.2 Parallelism analysis and scheduling challenge

One of the great interests of the hybrid methods in linear algebra is their coarse grain parallelism. Nevertheless, the parallelization of these methods is a complex and challenging work due firstly to the existence of their two main levels of parallelism, and then to the heterogeneity of the architectures being used as their execution support. The first level parallelism is that one inter co-methods constituting a hybrid method. The second level is the parallelism intra co-method which can be exploited according to a data parallel, message passing or multithreading programming model. We concentrate here on the inter co-method parallelism.

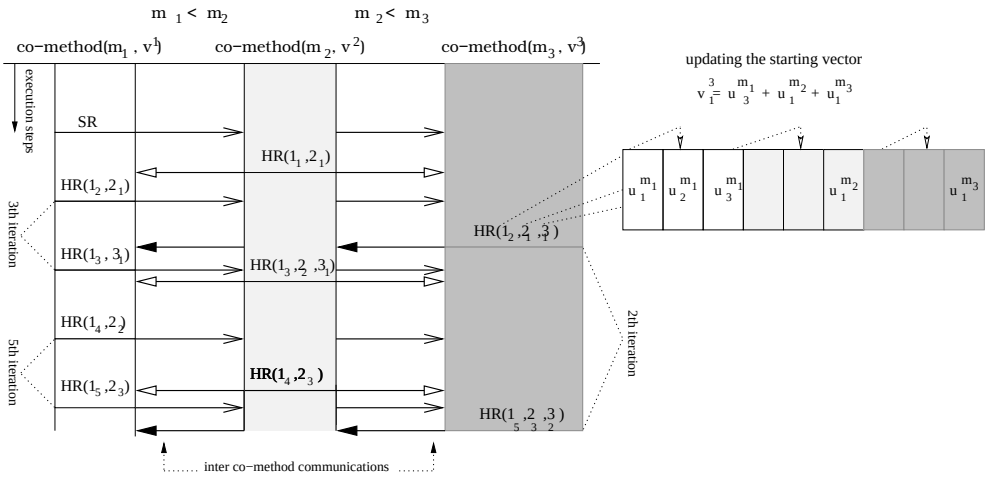

Fig. 1. A hybrid computation using $\ell = 3$ co-methods to compute an eigenpair (λ^m, u^m) of the matrix A. The co-method with the subspace size m_i and the initial guess v^i is denoted by co-method(m_i, v^i). HR and SR represent the hybrid and simple restarts.

Hybrid methods are well adapted for Grid computing. Different parallel components are just to be distributed to different resources. Nevertheless, the inter-component communications are asynchronous and difficult to be represented in a workflow model. Moreover, convergence detection introduces the necessity of interruption barriers between components. We will extend the workflow programming model to allow such asynchronous algorithms based, for example, on POP-C++ programming model [10]. Moreover, the components have to be simultaneously executed in order to collaborate. We exemplify this scheduling problem in the next subsection.

3.3 Scheduling hybrid methods: a basic example

Let us consider the same example described in Table 1. We suppose now to have three components which have to collaborate. Each component will use all available nodes on the allocated resource to maximize intra-parallelism work and therefore, to minimize iteration duration. We assume that the library L_2 is needed in order to run this collaborative work; all resources (R_1, R_2 and R_3) are thus admissible for running the components.

In a collaborative work, we have, as described in the cost model, simultaneity relations expressing the need to make all components running at the same time. In this example, those relations are $C_1 \simeq C_2$ and $C_2 \simeq C_3$.

A possible schedule for this hybrid method can be done as described in Table 4.

The metascheduler has relaxed the constraints of simultaneous starting time and proposes to wait until time 7 for taking benefit of the low cost period of

comp	resource	#proc	#start	#end	cost
C_1	R_1	12	6	15	3000
C_2	R_2	4	7	15	480
C_3	R_3	8	6	15	2000

Table 4. Example of schedules of co-methods in hybrid methods

R_2. C_1 and C_3 could begin a bit earlier without collaborating with C_2 at first. Maybe the use of only two co-methods during 1 time unit at first is not profitable. However, it is possible to set another schedule with all co-methods starting at time 7.

In the next section, we describe a programming environment well adapted to develop such hybrid methods on the Grid.

4 A candidate programming environment for developing hybrid methods on the Grid: POP-C++

4.1 Overview

The *POP-C++ programming environment* has been built to provide Grid programming facilities which greatly ease the development of parallel applications on the Grid. Figure 2 presents the layers of the POP-C++ architecture. The architecture supports the Grid-enabled application development at different levels, from the programming language for writing applications to the runtime system for executing applications on the Grid.

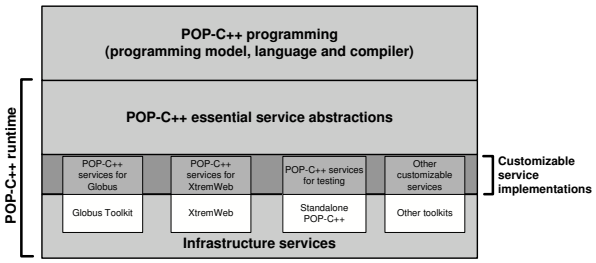

Fig. 2. The layered architecture of POP-C++ system

The POP-C++ runtime system consists of the infrastructure service layer managed by some Grid toolkits (e.g. Globus Toolkit or Unicore), the POP-C++ service layer to interface with the Grid infrastructures, and the POP-C++ essential service abstractions layer that provides a well defined abstract interface for the programming layer to access low-level services such as the resource discovery, the resource reservation or the object execution. The resource discovery and

reservation can be implemented using our proposed scheduling policy. Details of the POP-C++ runtime are described in [9].

POP-C++ programming, on top of the architecture, is the most important layer that provides necessary supports for developing Grid-enabled object-oriented applications based on the *parallel object model*.

4.2 POP-C++ programming model

The original *parallel object* model used in POP-C++ is the combination of powerful features of object-oriented programming and of high-level distributed programming capabilities. The model is based on the simple idea that objects are suitable structures to encapsulate and to distribute heterogeneous data and computing elements over the Grid. Programmers can guide the resource allocation for each object by describing their high-level resource requirements through the *object description*. The object creation process, supported by the POP-C++ runtime system, is transparent to programmers. Both inter-object and intra-object parallelism are supported through various original method invocation semantics. We intend to exploit POP-C++ objects and their descriptions to define components of hybrid methods and their timing constraints. Inter-object communications will be exploited for asynchronous inter co-methods communications.

The POP-C++ programming language extends C++ to support the parallel object model with just few new keywords for parallel object class declarations. Details of POP-C++ programming model are described in [8, 10]. With POP-C++, writing a Grid-enabled application becomes as simple as writing a sequential C++ application.

4.3 Parallel objects to capture components

One difficulty to develop and to deploy the component-based workflow model on the proposed scheduling system is the way to integrate resource requirements into each component. POP-C++ can help resolve this difficulty through its *object description* that allows programmers to describe their high level resource requirements such as the number of CPUs, the computing performance, the network bandwidth, etc. Although components can be implemented using any programming language, they are, in essence, very similar to POP-C++ objects. Nevertheless, the advantage of POP-C++ components is the ability to deduce all resource requirements of the components from their internal parallel structures. The proposed scheduling approach is well adapted to components written in POP-C++ but further study needs to be conducted in order to allow the POP-C++ compiler to automatically generate resources requirement of components.

5 Conclusion

In this paper, we have presented the problematic of scheduling intelligently some particular workflows on Grids. We have focus on a particular class of hybrid itera-

tive methods, which present collaborative asynchronous relations between coarse components. We have described a proposal for a generic cost model which can be a basis for the negotiation between a user and a metascheduler. We have investigated the feasibility of using the object-oriented programming environment POP-C++ for implementing and scheduling such hybrid methods on a computational Grid. Work is under way to implement and schedule an hybrid iterative method using the cost and evaluation models described in this paper and the programming environment POP-C++.

Acknowledgements

The authors wish to thank the European Commission for its support under the FP6 Network Of Excellence CoreGRID, which has permitted this joint work, and express their gratitude to the referees for their valuable comments.

References

1. J.M. Bahi, S. Contassot-Vivier and R. Couturier: *Asynchronism for iterative algorithms in a global computing environment.* HPCS'02, IEEE Computer Society Press, 2002.
2. M. Aldunicci, S. Campa, M. Coppola, M. Danuletto, D. Laforenza, D. Puppin, L. Scarponi, M. Vanneschi, C. Zoccolo: *Components for high-performance grid programming in GRID.IT, in Components models and Systems for grid applications* V. Getov and T.Kielmann Eds, Springer, 2005, 19–38.
3. N. Emad, S. G. Petiton and G. Edjlali: *Multiple explicitly restarted Arnoldi method for solving large eigenproblems.* SIAM Journal on scientific computing SJSC, Volume 27, Number 1, pp. 253-277(2005)
4. R. Gruber, P.Volgers, A. De Vita, M. Stengel, T-M Tran: *Parameterisation to tailor commodity clusters to applications.* Future Generation Comp. Syst., 19,1,111-120,2003
5. H. He, G. Bergère and S. G. Petiton: *A Hybrid GMRES-LS-Arnoldi method to accelerate the parallel solution of linear systems .* Future Generation Comp. Syst., 19,1,111-120, 2003
6. V. Keller, K. Cristiano, R. Gruber, T-M Tran, P. Kuonen, P. Wieder, W. Ziegler, S. Maffioletti, N. Nellari, M-C Sawley: *Integration of ISS into the VIOLA Meta-scheduling Environment.* Computer and Mathematics with applications, 2005
7. M.S. Sadiq, Y. Habib: *Iterative computer algorithms in engineering: solving combinatorial optimization problems.* Wiley, 2000
8. T.A Nguyen, P. Kuonen: *ParoC++: A Requirement-driven Parallel Object-oriented Programming Language.* Proc. of the 8th International Workshop on High-Level Programming Models and Supportive Environments/IPDPS, 2003
9. T.A. Nguyen: An Object-oriented model for adaptive high performance computing on the computational Grid. PhD thesis, Swiss Federal Institute of Technology-Lausanne, 2004
10. T.A Nguyen, P. Kuonen: *Programming the Grid with POP-C++.* Future Generation Computer Systems, submitted 2005