HAL
archives-ouvertes.fr

Dynamic adaptation for Grid computing

Jérémy Buisson, Françoise André, Jean-Louis Pazat

▶ **To cite this version:**

Jérémy Buisson, Françoise André, Jean-Louis Pazat. Dynamic adaptation for Grid computing. European Grid Conference, Feb 2005, Amsterdam, Netherlands. pp.538, 10.1007/11508380_55 . hal-00498845

HAL Id: hal-00498845

https://hal.archives-ouvertes.fr/hal-00498845

Submitted on 8 Jul 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Dynamic adaptation for Grid computing

Jérémy Buisson[1], Françoise André[2] and Jean-Louis Pazat[1]

[1] IRISA/INSA de Rennes, Rennes, France
[2] IRISA/Université de Rennes 1, Rennes, France

Abstract. As Grid architectures provide resources that fluctuate, applications that should be run on such environments must be able to take into account the changes that may occur. This document describes how applications can be built from components that may dynamically adapt themselves. We propose a generic framework to help the developpers of such components. In the case of a component that encapsulates a parallel code, a consistency model for the dynamic adaptation is defined. An implementation of a restricted consistency model allowed us to experiment our ideas.

1 Introduction

Grid architectures differ from classical execution environments in that they are mainly built up as a federation of pooled resources. Those resources include processing elements, storage, network, and so on; they come from the interconnection of parallel machines, clusters, or any workstation. One of the main properties of these resources is to have changing characteristics even during the execution of an application. Resources may come and go; their capacities may vary during the execution of the applications. Moreover, resources may be allocated then reclaimed and reallocated as applications start and terminate on the Grid. Thus, resource usage by applications cannot be static; neither can changes in resource allocation be considered as faults. Grid-enabled application designers must keep in mind that resources and resource management are highly dynamic within Grid architectures.

Dynamic adaptation is a way to support evolving execution environments. It aims at allowing applications to modify themselves depending on the available resources. The key idea is that when the environment changes, the application should also change to fit with it. To react softly, the application can negotiates with its execution environment about resources, instead of undergoing the decisions of the execution environment. Because taking adaptation into account should not burden too much the application developpers, it is necessary to provide them adequate frameworks and mechanisms. That is our main research objective.

Section 2 makes a tour of existing researches in areas close to adaptation. Section 3 exposes how we model the dynamic adaptation of an application. Section 4 shows the architecture we are currently building as a support to make adaptable applications. Section 5 describes the consistency model we introduced

for the adaptation of components that encapsulate parallel codes. Section 6 presents the state of the experiments we have done with our consistency model.

2 Adaptation through existing works

Several projects use the word adaptation to describe themselves. As this section presents, all of those projects have different views of what adaptation is, what it consists in and why it should be used. Basically, the only common point is that adaptation consists in changing some things in applications. This asks four major questions: why adaptation should be done (section 2.1); where is it done in the execution (section 2.2); how can it be done (section 2.3); when should it be done (section 2.4). As the section shows, the several projects that exist have different answers to these four questions.

2.1 Goal of the adaptation

With existing projects, two main views of what the goal of adaptation could be opposed. In [1], the Grid.It project and its ASSIST [2] programming model present adaptation as a way to achieve a specified level of performance. An application should modify its structure or change the resources it has allocated when performance contracts are not satisfied.

On the other hand, projects such as SaNS [3], GrADS [4] and PCL [5] consider that applications should adapt themselves to optimize themselves. Adaptation is a way for application to provide a "best effort" strategy. In this case, the application chooses the best implementation given the allocated resources and the properties of input data.

2.2 Location of the adaptation

Many projects such as SaNS, GrADS and GrADSolve [6] do the adaptation upon invokation: when a function, method, procedure is called, the best algorithm is chosen. Although this is not as static as the automatic optimization such as for ATLAS [7], adaptation is not possible once the adaptable software has started its execution. This approach is realistic only if the adaptable softwares are fine-grained enough. In the case of SaNS, the targeted softwares are libraries of numerical kernels. In this case, it is sufficient to adapt only at invokation time.

Some projects such as Grit.It and PCL allows dynamic adaptation (namely, adaptation of a software that is executing). This is necessary when the execution time of the adaptable software is high. Moreover, the PCL project defines in [8] a consistency model for the adaptation of parallel codes.

2.3 Means of the adaptation

When the adaptation is done exclusively at invokation time, adaptation is simply the selection of one implementation of several that are available. This is what is done with SaNS and GrADSolve.

In the case of dynamic adaptation, this is not sufficient. Adaptation either involves parameterizable softwares or reflexive programming. For example, within the ASSIST model of the Grid.It project, the *parmod* skeleton exposes several parameters such as the degree of parallelism that can be changed dynamically; the PCL project defines a framework for reflexive programming in distributed and parallel applications. Dynamic aspect weavers (such as Arachne [9]) may also be used as an alternative to reflexive programming to dynamically adapt the software.

2.4 Decisions to adapt

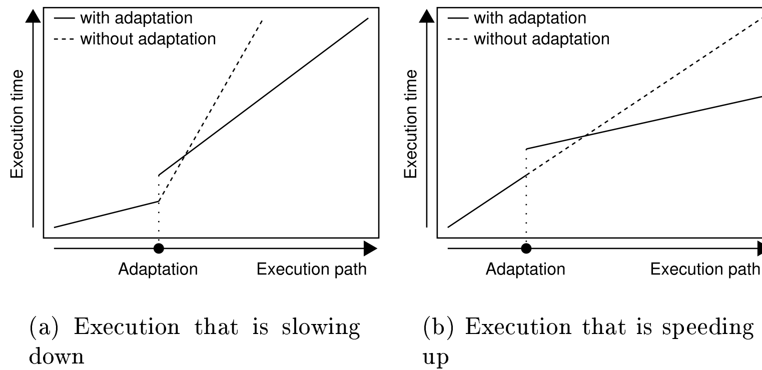
Several approaches can be used to decide when a software should adapt itself and what it should do. Within the ASSIST model, since adaptation is a way to achieve the contracted performance level, adaptation is mostly triggered from feedback control. On the other side, within the PCL project, the trigger is the reception of external events generated by external monitors. The projects SaNS and GrADSolve use performance models and a database of empirical measures to choose the right implementation to use.

3 Model of dynamic adaptation

Our model for dynamic adaptation considers that dynamic adaptation should occur in order to maximize the adequation between the application and its execution environment. Namely, this means that the purpose of dynamic adaptation is to optimize the application whenever its execution environment changes. To do so, it may use whatever means the developper is willing to.

The figure 1 shows the two kinds of curves for the progress of an adaptive software. The two cases correspond to an adaptation that causes the execution to slow down 1(a) and one that makes the application accelerates 1(b). The discontinuity reflects the cost of the adaptation. Although the adaptation optimizes the application, the application may slow down. Indeed, this occurs when the execution environment reclaims some resources. Even if the application slows down, it is optimized compared to its execution if it has not been adapted: in such a case, the application could even have crashed.

The application can adapt itself anywhere in the execution path. It can be either at a past state or at a state in the future. The figure 2 shows how the overall completion time conceptually behaves depending on the current state and the one at which the adaptation is done. This curve directly translates that the adaptation optimizes the application. Thus, in the future, the sooner it is done, the better the completion time is. It reduces the time during which the application is suboptimal. By symmetry, in the past, the earlier in the application lifetime the adaptation is done, the bigger the computations to be redone are. Adapting "just after now" is better than "just before now" because of the overhead of restoring a past state.



(a) Execution that is slowing down

(b) Execution that is speeding up

Fig. 1. Progress of an adaptive software

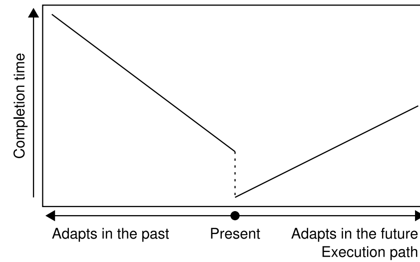


Fig. 2. Completion time depending on the state at which the adaptation is done

Adapting in the past has the advantage over adapting in the future that the adaptation can occur immediatly once the decision has been made. Thus, there is no transition during which the execution is suboptimal. Moreover, it does not require the prediction of a point in the future of the execution path, which is an undecidable problem in the general case.

On the other hand, adapting in the future does not require any instruction to be reexecuted; neither it has the overhead due to checkpoint-taking.

4 Architecture of adaptable software

Applications are considered to be built as assemblies of software components. Each of those components can encapsulate sequential or parallel codes. Moreover, each component can adapt itself. To do so, the component is supported by a framework. Indeed, it appears that some of the mechanisms involved in adaptation are independant of the component itself.

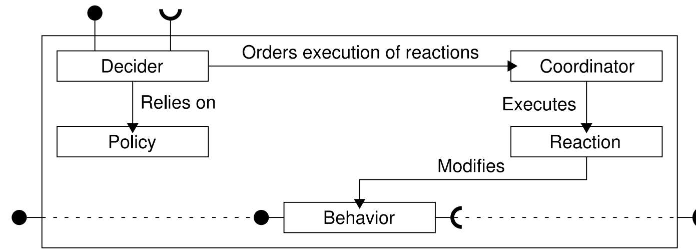


Fig. 3. Overall architecture of an adaptable component

4.1 Architecture of an adaptable component

The figure 3 shows the architecture of a parallel component. Five major functional boxes have been identified that lie in three parts of an adaptable component built with our adaptation framework.

- Functional part of the component.
  - **Behavior.** This is the implementation of the services provided by the component. An adaptable component is allowed to contain several behaviors that are alternative implementations.
- Component-specific part of the adaptation framework.
  - **Reaction.** This is a code that modifies the behavior that is being executed by the component. A component can include several reactions. The reactions can change some parameters of the behavior; replace the behavior with another one; modify the behavior with the help of reflexive programming or dynamic aspect weaving.
  - **Policy.** This provides all the necessary information that are specific to the component to make decisions concerning the adaptation. It eddicts on which events the component should adapt and which reaction it should use.
- Component-independant part of the adaptation framework.
  - **Coordinator.** The coordinator is responsible for choosing where the reaction is going to be executed within the execution path of the behavior. The possible locations are called candidate points; the chosen one is the adaptation point.
  - **Decider.** The decider decides when the component should adapt itself and chooses which reaction should be executed. To do so, it relies on the information given by the policy.

The coordinator and the decider both make decisions regarding the adaptation. However, they encapsulate separate concerns. The decider encapsulates the goal of the adaptation and the criterium that is going to be optimized; whereas the coordinator focuses on the mechanisms for enforcing the consistency of the dynamic adaptation.

4.2 Scenario of the adaptation of a component

From time to time, the decider decides that the component should adapt itself with one reaction. This may come from an external event (specified in the policy of the component); this may be a spontaneous decision (as a result of feedback control for example). Once the decision to adapt is made, the decider orders the coordinator to execute the chosen reaction. Then, the coordinator chooses one candidate point in the execution path of the behavior. It also starts to monitor the execution of the behavior. When it reaches the adaptation point, the coordinator suspends the execution of the behavior and gives the execution control to the reaction. Once the reaction has finished, the behavior resumes its execution.

In addition, the behavior of a component can be a parallel code. In this case, the candidate points and the chosen adaptation point are global and represent global states. A global point is composed of one local point for each concurrent thread of the parallel behavior. Local points are identified by both their name in a model of the functional code and the indices in the iteration spaces.

The decider relies on the policy to make its decisions. The policy may contain an explicit set of rules: this can help the decider to choose the events to subscribe to. The policy may also contain performance models to help to choose the right reaction. Those performance models are parameterized with the content of the contracts describing the quality of the used services and resources.

5 Consistency of the adaptation within parallel components

When the component encapsulates a parallel code, the coordinator must choose a global point. However, the result of the execution of the adapted component must be semantically equivalent to the one of the component that does not adapt itself. Thus, the chosen point should be consistent with respect to a given property that garanties this semantic equivalence. The adaptation is said consistent if the reaction is executed from such a point.

A simple example illustrates this: if the component does not dead-lock in normal executions, it must not dead-lock when it adapts itself. Thus, choosing the point at which the reaction can be executed requires to be aware of the communications that occur in the behaviors and reactions of the component.

5.1 Consistency model

The global point is R-consistent if and only if it satisfies the R relation. It is a n-ary relation if the parallel behavior is composed of n concurrent threads. This relation is defined over the n sets of local points. It is specific to each component.

In our example about dead-lock, this R relation encapsulates the required knowledge about communications within the component.

5.2 Classes of parallel components

Depending on the properties of the R relation, the parallel components can be classified. There is four major classes of parallel components.

SPMD components. If the R relation is id the identity, a global point is consistent if all the threads are locally at the same point. Regarding to the points, this means that all the threads share the same set of points. This case corresponds to the SPMD class of parallel applications.

Quasi-SPMD components. We consider that the R relation holds the following property:

$$\begin{aligned}
&(R(p_1, p_2, \ldots, p_n) \wedge R(q_1, q_2, \ldots, q_n)) \\
&\Rightarrow (p_1 \prec q_1 \wedge p_2 \prec q_2 \wedge \ldots \wedge p_n \prec q_n) \\
&\vee (q_1 \prec p_1 \wedge q_2 \prec p_2 \wedge \ldots \wedge q_n \prec p_n) \vee (q_1 = p_1 \wedge q_2 = p_2 \wedge \ldots \wedge q_n = p_n)
\end{aligned}$$

The $\prec$ symbol denotes the strict "precede" relation over the sets of local points. In the execution paths, this is a total order relation.

If this property is satisfied, the local points can be renamed such that R becomes the identity relation. Thus, the component behaves like SPMD components.

Synchronous MPMD components. We suppose that the R relation satisfies the following property:

$$\begin{aligned}
&(R(p_1, p_2, \ldots, p_n) \wedge R(q_1, q_2, \ldots, q_n)) \\
&\Rightarrow (p_1 \preceq q_1 \wedge p_2 \preceq q_2 \wedge \ldots \wedge p_n \preceq q_n) \vee (q_1 \preceq p_1 \wedge q_2 \preceq p_2 \wedge \ldots \wedge q_n \preceq p_n)
\end{aligned}$$

The $\preceq$ symbol denotes the reflexive "precede" relation over the sets of local points. In the execution paths, this is a total order relation.

In this case, the global points are still totally ordered in the execution path by a "precede" relation. This reflects the synchronization of the threads within the component. However, a local point can participate to several global points. Those components can be rewritten as quasi-SPMD components by duplicating such local points. This transformation restricts the consistency model.

Pessimistic parallel discrete event simulators belong to this class of components.

Asynchronous MPMD components. If none of the preceding properties is satisfied, the "precede" relation over global points in execution path is not a total order relation.

This class of components includes in particular master-slaves codes.

5.3 Comparison with other consistency models

The model defined by PCL in [8] says that the adaptation is consistent if all the threads reach the i-th point (same i for all the threads) in the execution path. Although it seems similar, this is not equivalent to our *id*-consistency. Indeed, our model identifies points by name (identifier in the model of the code augmented with indices in iteration spaces) whereas PCL explicitly uses the rank of the point in the execution path.

Thus, our model accepts that the threads have different dynamic behaviors. The threads are not expected to execute the same number of iterations or to choose the same branch of conditional instructions. On the other hand, PCL supposes that all the threads remain in sync.

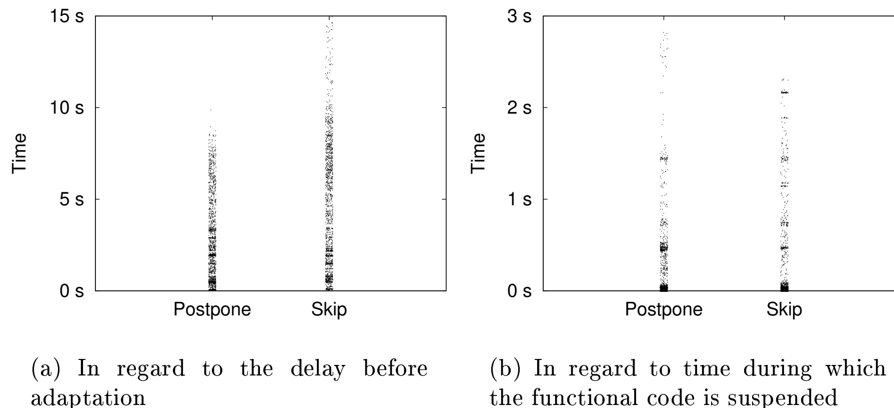
6 Realisation

By the time, we have designed and implemented an algorithmic solution for the coordinator. Our work restricts the model to the *id*-consistency in the case of SPMD parallel components. The candidate global points are exclusively looked for in the future of the execution path of the threads.

We have chosen to restrict to candidate points in the future. We worked around the impossibility of predicting the next point in the future of the execution path by introducing several strategies: the "postpone" strategy delays the prediction until the conditional instruction is executed; the "skip" strategy ignores the candidate points within the branches of the conditional instruction.

Figure 4 compares these two strategies. This experiment has been done with the NPB 3.1 [10] FFT code on a 4 PCs cluster running PadicoTM [11], which permits us to mix several middlewares such as MPI and CORBA within a single application. Only the coordinator has been implemented: the reaction is empty (no real adaptation); the decider is simulated by a trigger provided to the user. The encapsulation within a component has not been done as it has no influence for this experiment if we consider that the whole FFT code is within a single component.

Each dot of figure 4 represents one trigger of the adaptation. It appears on figure 4(a) that the "postpone" strategy generally chooses adaptation points that come sooner than the "skip" strategy. On the other hand, the figure 4(b) shows that the "postpone" strategy suspends more frequently the functional code than the "skip" strategy. Indeed, we observe that most of the experiments with the "skip" strategy causes no suspension (high density of dots at 0 s), whereas the functional code is frequently suspended with the "postpone" strategy (high density of dots up to about 0.5 s). These two observations exhibit the trade-off between the precision of the prediction and the risk of suspending the functional code.



(a) In regard to the delay before adaptation

(b) In regard to time during which the functional code is suspended

Fig. 4. Comparison of the “postpone” and the “skip” strategies

7 Conclusion

In this paper we have shown that adaptation of parallel components can be achieved using our framework. Such an adaptation process needs to develop a consistency model and an algorithm to enforce this consistency before adaptation: the *id*-consistency model we developped shows the relevance of our consistency model for the adaptation of SPMD parallel codes.

We have shown through the FFT example that there is a trade-off between the precision of the choice of the point (best theoretical result) and the risk of uselessly suspending the execution of the functional code.

Futur works around the coordinator will consist in extending our implementation of the consistency model for non-SPMD components.

We plan to fully implement our framework, including a smart decider. This decider will be organized as a rule-based system. We will study how to represent usefull information for decision-making, such as states or changes of the environment. The representation of the adaptation policy have also to be defined. For this work we plan to rely on previous works on monitoring such as Delphoi [12] or SAJE/RAJE [13] and on our previous works on adaptation for mobile computing [14].

Another direction of our work is the study of the adaptation of assemblies of components for complete Grid applications. It is clear that in many applications, one component will not be able to adapt itself without taking into account the other components of the application. The role of the decider will be extended to include negotiation with deciders of other components of the application.

References

1. Aldinucci, M., Campa, S., Coppola, M., Danelutto, M., Laforenza, D., Puppin, D., Scarponi, L., Vanneschi, M., Zoccolo, C.: Components for high performance grid programming int the grid.it project. In: Workshop on Component Models andd Systems for Grid Applications. (2004)
2. Aldinucci, M., Coppola, M., Danelutto, M., Vanneschi, M., Zoccolo, C.: Assist as a research framework for high-performance grid programming environments. Technical Report TR-04-09, Università di Pisa, Dipartimento di Informatica, via F. Buonarroti 2, 56127 Pisa, Italy (2004)
3. Dongarra, J., Eijkhout, V.: Self-adapting numerical software for next generation application (2002)
4. Kennedy, K., Mazina, M., Mellor-Crummey, J., Cooper, K., Torczon, L., Berman, F., Chien, A., Dail, H., Sievert, O., Angulo, D., Foster, I., Gannon, D., Johnsson, L., Kesselman, C., Aydt, R., Reed, D., Dongarra, J., Vadhiyar, S., Wolski, R.: Toward a framework for preparing and executing adaptive grid programs. In: Proceedings of NSF Next Generation Systems Program Workshop (IPDPS). (2002)
5. Adve, V., Lam, V.V., Ensink, B.: Language and compiler support for adaptive distributed applications. In: ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001), Snowbird, Utah (2001)
6. Vadhiyar, S., Dongarra, J.: GrADSolve: RPC for high performance computing on the grid. In: Euro-Par 2003: Parallel Processing. Volume 2790. (2003)
7. Whaley, R.C., Petitet, A., Dongarra, J.J.: Automated empirical optimizations of software and the ATLAS project (2000)
8. Ensink, B., Adve, V.: Coordinating adaptations in distributed systems. In: 24th International Conference on Distributed Computing Systems. (2004) 446–455
9. Ségura-Devillechaise, M., Menaud, J.M., Muller, G., Lawall, J.: Web cache prefetching as an aspect: Towards a dynamic-weaving based solution. In: Proceedings of the 2nd international conference on Aspect-oriented software development, ACM Press (2003) 110–119
10. : Nas parallel benchmark. (http://www.nas.nasa.gov/Software/NPB/)
11. : PadicoTM. (http://www.irisa.fr/paris/Padicotm/welcome.htm)
12. Maassen, J., van Nieuwpoort, R.V., Kielmann, T., Verstoep, K.: Middleware adaptation with the delphoi service. In: AGridM 2004 - Proceedings of the 2004 Workshop on Adaptive Grid Middleware, Antibes Juan-Les-Pins, France (2004)
13. Guidec, F., Sommer, N.L.: Towards resource consumption accouting and control in java: a practical experience. In: Workshop on Resource Management for Safe Language, ECOOP 2002, Malaga, Spain (2002)
14. Chefrour, D., André, F.: Développement d'applications en environnements mobiles à l'aide du modèle de composant adaptatif aceel. In: Langages et Modèles à Objets LMO'03. Actes publiés dans la Revue STI. Volume 9 of L'objet. (2003)