Grid resource allocation and reallocation for adaptable applications

Report of CoreGRID REP 9 (CR14 INRIA - CR06 TUD)

Jérémy Buisson and Dick Epema

15 january – 11 may 2007

1 Introduction

This report summarizes the activities and main achievements that have resulted in the context of the CoreGRID network of excellence from REP 9 on resource allocation for adaptable applications. During this 4 months program, Jérémy Buisson (postdoc, CR14 INRIA) has collaborated with Dick Epema (associate professor, scientific contact in the hosting institute, CR06 TUD), Hashim Mohamed (PhD student, CR06 TUD) and Ozan Sonmez (PhD student, CR06 TUD).

The program has been proposed in the context of making grid applications better use resources; despite resource availability varies frequently in platforms like grids. While dynamic adaptability makes applications handle varying resource allocations, this ability is useless as long as infrastructures and grid schedulers are not able to produce such allocations. Application adaptability is thus commonly not supported by most of the grid infrastructures. This is the issue that has been studied during the exchange program, of which the results are reported here. We have focused on making Koala, the grid scheduler developed by partner CR06 TUD, schedule malleable jobs. We have also investigated how applications based on Dynaco, the adaptability framework of partner CR14 INRIA, can take into account decisions made by the system-wide scheduler.

The report is structured as follows. Section 2 describes the Koala grid scheduler and the Dynaco adaptability framework, which have been previously proposed by the partners involved in the program. Section 3 summarizes the results obtained during the program. Section 4 concludes the report with the implementation status of proposals that result from the program. It also relates those proposals with regard to the other activities of CoreGRID and gives some future directions.

2 Background

With the emergence of grid computing, scientists and users are given large-scale platforms in order to execute resource intensive applications. However, grids still challenge researchers to propose adequate infrastructures, programming models and tools in order to make grid applications execute efficiently. This section summarizes the previous contributions of partners CR06 TUD and CR14 INRIA to those challenges. Those contributions are the starting point of the program.

2.1 Grid scheduler and resource manager

Grid infrastructures [13, 17, 18] aim at giving access to resources in a secure and fair manner. Like Globus [13] for instance, in addition to creating processes on users' behalf thanks to local resource managers, infrastructures enforce access control, authenticate users and manage data movement and replication between organizations that participate to grids. Grid-level schedulers, who indicate to each application which resources they should use and when they should execute, contribute to balance the load over the whole platform and to increase fairness.

The KOALA [20] grid scheduler has been developed by partner CR06 TUD. Grid scheduling distinguishes from classical scheduling problems (for instance in comparison to clusters and shared-memory multiprocessors) in that the scheduler does not have full control of the resources it allocates. Indeed, users are allowed to bypass the grid infrastructure when accessing their local resources. KOALA aims at dealing with the resulting volatility of resources, as well as with co-allocating resources from several uncoordinated local resource managers. To do so, it monitors resource availability in all of the grid organizations. When enough resources are available, it selects some jobs from its queues for execution. In case no reservation mechanism can be used, if some of the allocated resources are taken by concurrent users before being obtained, jobs are re-queued.

In the job model of KOALA, jobs are composed of components. Components are primitive sub-jobs that are expected to run concurrently, at the same time and for the same duration. Each component is allocated resources from a single local resource manager. But the components of a single job may be distributed over several local resource managers. The specification of components can be more or less constrained: it can impose a given local resource manager or not; it can request a statically fixed amount of resources or give a range of possible values. Whatever the case, resources are allocated for the whole execution time of the job.

In order not to be tied to a specific class of applications, KOALA exhibits a clear separation between the scheduler and the startup procedure of applications. From the architectural point of view, KOALA defines *runner*s, which are responsible to translate user-provided requests into a generic request language (based on GLOBUS RSL) and to start the application. Runners have also to reserve, claim then release allocated resources thanks to an underlying grid infrastructure such as GLOBUS. Several runners have been contributed in order to execute GLOBUS/DUROC, MPICH-G2, IBIS and "classical" applications.

In addition to being a research prototype, KOALA has been successfully deployed on the DAS [1] platform. It operates as a production grid-level scheduler in DAS-2 and DAS-3 on top of SGE and the GLOBUS infrastructure.

2.2 Grid programming models and adaptable components

According to the results of many research groups [3, 5, 6, 16, 22, 23], software component technologies appear to be valuable in order to deal with the complexity of grid applications. Decomposition of applications gives a natural granularity for the use of distributed resources; while encapsulation fosters the reuse of existing components. In addition, usual paradigms (such as data-parallelism or skeletons) abstract the details of parallel implementations inside each component. Several models have been contributed in order to allow parallel implementations of components, such as GRID-CCM [23], ASSIST [3], CCA [5], PROACTIVE [6] and GCM [22]. As in addition resource availability varies frequently in grids, adaptability and autonomous computing techniques have been transposed to the context of grid components with DYNACO/AFPAC [9], ASSIST [4], GRADS [26] and GRIDWAY [15], which make components/applications adapt their resource consumption to the actual availability in grids. Grid applications are thus built as adaptable assemblies of components, which are themselves adaptable.

The DYNACO framework has been developed by partner CR14 INRIA in order to help developers design and implement adaptable components. As a complement to existing component models, DYNACO focuses exclusively on adaptability. The DYNACO framework enacts the separation of concerns between adaptability and other functionalities of adaptable components as it lays aside the implementation of the component it controls. Inheriting from our experience with MOLÈNE [24] and ACEEL [11] and benefiting from a previous collaboration with the university of Pisa [2], DYNACO decomposes the process of adaptability into specific functions: *observe* in order to detect whenever the environment evolves and to trigger the adaptation; *decide* what is the best suited strategy according to the collected measures; *plan* the actions that have to be executed in order to achieve the strategy; and *execute* those actions, taking care of the synchronization with application code. Those functions are gathered in an assembly of abstract FRACTAL [7] components. When developers design and implement an adaptable component, they are expected to materialize this assembly with concrete FRACTAL components. To do so, they can rely on

well-known generic engines: we have for instance successfully used rule-based expert systems and meta-heuristic optimizers. The DYNACO framework is open and does not restrict arbitrarily to any engine. Similarly, even if it is expressed as an assembly of FRACTAL components, the framework does not restrict the application to adhere that component model. Only adaptation actions have to be exposed thanks to FRACTAL components. Those components can implement application functionalities; they can also simply forward invocations out of the FRACTAL architecture of adaptability.

In our previous works [9], we have used DYNACO in conjunction with AFPAC [10], which synchronizes adaptation actions with the execution of parallel SPMD application codes, in order to make malleable several applications (including a FFT numerical kernel and an n body simulator). The instance of DYNACO monitors the collection of allocated resources and adapts the collection of used resources accordingly.

2.3 Disambiguation of the word "component"

The word "component" is used in three different contexts, which inherit from specific existing terminologies. For the sake of clarity, we emphasize the differences.

- In the context of KOALA, *job component*s result from the decomposition of jobs. They are the allocation units that are transmitted to local resource managers.
- In the context of DYNACO, the *adaptable component* is the scope of adaptability. As such, it denotes a decomposition unit of applications, which may enclose several scopes of adaptability. The actual nature of *adaptable component*s may be determined by the application programming model. For instance, they can be GCM components in the context of GCM.
- In the architecture of the DYNACO framework, *component*s are units of decomposition that adhere to the FRACTAL component model. Those *component*s are the building blocks of the DYNACO framework. In each instance of the framework, the controlled *adaptable component* is modeled by one or several FRACTAL *component*s that expose the adaptation actions.

3 Main achievements

During the program, we have extended KOALA in order to make it able to change resource allocation while jobs are active. We have also modified existing adaptable components in order to take into account the specificities of the DAS grid and we have designed a specific KOALA runner for DYNACO-based applications. At last, we have studied scheduling policies for malleable applications in the context of grid computing.

Figure 1 summarizes the overall proposed architecture. It emphasizes the design patterns that are reused (KOALA, DYNACO, AFPAC and GLOBUS) and how they are connected in the global architecture. The figure also highlights the focus of each of the following sections.

3.1 Managing malleable job in grid infrastructures

Most existing grid infrastructures, including GLOBUS, provide only with rigid jobs: the amount of allocated resources is constant for the whole lifetime of the job. Such management prevents applications malleability. In order to overcome this problem, we have considered two approaches.

The limitation can be worked around by a layer built on top of the grid infrastructure. That new layer implements malleable jobs as dynamic collections of rigid jobs. Several existing tools proceed in that way, such as CONDOR-G *glide-in* [14]. Leaving the grid infrastructure unmodified, this approach fits well legacy platforms. However, managing a collection of rigid jobs involves many interactions with the grid infrastructures (proportional to the size of malleable jobs). Consequently, the cost of managing malleable jobs in that way is excessively high, as for instance each interaction with the grid infrastructures requires to revalidate credentials and to check against access control

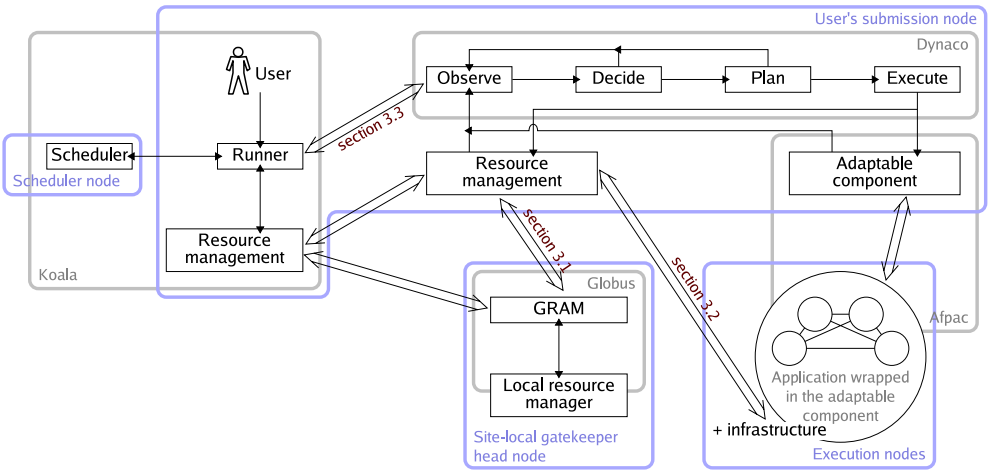

Figure 1: Overall architecture of an adaptable application that uses DYNACO and that is deployed into a grid operated by KOALA and GLOBUS. Single arrows show the bindings between elements within each design pattern. Double arrows connect elements that interface design patterns together.

rules. Furthermore, possible facilities of local resource managers for managing malleable jobs are masked by grid infrastructures.

Alternatively, the grid infrastructure can be updated in order to build malleable job management into it. Infrastructures such as GRIDKIT [12] provide such flexible management. GRIDKIT component frameworks allow administrators to configure (and dynamically reconfigure) the infrastructure: administrators can choose the components that implement each service. In addition, in order to anticipate forthcoming additional features, a domain-specific extensibility mechanism should be designed in the job model, as in OGSA-BES [21]. However, in both cases, job management code is still exclusively part of the infrastructure. If on the contrary users were invited to participate to the implementation of job management, new features could be provided without any further need to update and redeploy grid infrastructures. To do so, users should be allowed to attach their job management code to job descriptions, such that it can be plugged at runtime into the infrastructure. Users are thus able to configure and extend the grid infrastructure on a per-job basis in order to fit their actual needs.

This issue is further discussed in [BAP07], which has been submitted to Grid 2007.

3.2 Coupling resource management with adaptable applications

In order to couple resource management with adaptable applications, two main options can be considered: adaptation can be done according to the collection of resources in the malleable job; or according to the collection of resources that are available to the malleable job.

In the former case (adapting to resources in the job), adaptability and its implementation are not modified when coupling with the resource management system. Resource management is entirely outsourced out of the application. However, applications may obtain resources that they are unable to use, unless a protocol is designed for negotiating resource allocations. Indeed, the grid infrastructure cannot be assumed to know about the resource constraints for any application.

In the latter case (adapting to the available resources), adaptability has to be extended in order to claim and release resources explicitly. Feedback can be used in order that resource management operations get separated from the rest of adaptability, resulting in better modularization. Adaptation is then split into two sub-adaptations: changes in resource availability trigger the adaptation of the malleable job; changes of the size of the malleable job trigger the adaptation of the application. Therefore, extensions of adaptability consist exclusively in adding new rules in the decision

procedure, new plans and new adaptation actions, which all lay aside of the unmodified previously existing ones. These are the reasons why we have chosen this solution.

In addition, the effective use of resources by application code is separated from their procurement into the malleable job. In order to allow the application to access resources in the malleable job, an overlay layer is installed when resources are obtained from the grid infrastructure. In addition to the collection of resources that is maintained by the malleable job, that layer provides with a mechanism to later connect to resources (i.e. remote process creation).

3.3 Coupling adaptable applications with Koala

In order to execute adaptable applications that use DYNACO with the KOALA scheduler, we have designed a specific KOALA runner. In the overall architecture, the runner is given several roles. It retains from the KOALA design its role of translating users' application-specific requests into the generic RSL. In so doing, the runner maintains a mapping between job components (specified in the RSL request that gives application's resource requirements) and adaptable components (specified in the FRACTAL software architecture that describes the application).

As a DYNACO monitor component, the runner forwards *grow* and *shrink* messages from the scheduler throughout the adaptability framework. Targets of events are identified thanks to the above mapping. The runner also catches results of adaptations in order to generate acknowledgements to the scheduler.

In addition, the runner hosts the control part of the application in a centralized manner. It includes component management (as provided by FRACTAL) as well as instances of the DYNACO framework. Unlike the other runners, this one is thus dynamically extended with application code; and resource management (e.g. interface with GLOBUS) is delegated to that code as stated in Section 3.2.

3.4 Grid scheduling policies for malleable applications

The KOALA grid scheduler has been extended such that it can benefit from malleable applications. In the scheduling policies, several options have been identified. They favor either already executing jobs or pending jobs. The scheduler reacts to the following four conditions:

- *When a job is waiting for placement.* Two cases may occur. If enough resources are available, then the job is placed. Otherwise, the job is queued (favoring executing jobs), or *shrink* messages are sent to make room for the job (favoring pending jobs).
- *When some resources are available.* Available resources are distributed over executing malleable jobs and pending jobs. Depending on the order in which they are considered, jobs in one category are favored over jobs in the other one.
- *When a job requests for growing.* If enough resources are available, growth is granted. Otherwise the scheduler attempts to shrink other malleable jobs before denying the growth.
- *When a failure occurs.* The scheduler let applications handle failures on their own. The scheduler gives no guaranty on the quality of the resources.

No distinction is made between rigid and malleable jobs. Shrinking jobs occurs only when pending jobs are favored over the executing ones.

In addition to that outlined scheduling algorithm, two policies have been identified where several alternatives can be considered. A placement policy is inherited from the initial KOALA scheduler. It states how to choose a cluster for each job component. According to the expertise of partner CR06 TUD, the following algorithms are relevant: worst fit [8]; close to file [19]; cluster minimization [25]; flexible cluster minimization [25]. When the scheduler decides the amount by which jobs are grown or shrunk, we propose either that it performs equipartition, or that it fills jobs as much as possible in a given order.

Regarding job specification, we have proposed to require the minimal amount of resources that each job must be guaranteed to obtain. The scheduler can increase jobs arbitrarily up to an explicit upper-bound. In that range, the scheduler is allowed to select any value without any application-specific constraint. If necessary, applications can still release extra resources that they are unable to use. This mechanism is the same as when applications voluntarily release some of their resources, while it still avoids asking the resource management system for useless resources.

More details, including an empirical analysis of the behavior of the policies, can be found in [BSMLE07], which has been submitted to the Cluster 2007 conference.

4 Conclusion

4.1 Relation to initial goals

During the program, we have extended the protocol between the scheduler and KOALA runners in order to allow modifications of resource allocations. We have proposed an algorithm in order to reschedule applications dynamically, which has been implemented in the KOALA scheduler. A specific KOALA runner has been developed for DYNACO-based applications. Several variants of the scheduler algorithm have been compared empirically with regard to metrics such as utilization, response time and consumed computational power. Results of the program have been submitted as a joint paper [BSMLE07] to Cluster 2007. Therefore, initial goals have been met.

4.2 Relation to CoreGRID activities

Results of the program relates to the activities of CoreGRID in several ways. GCM, which is the component model that WP3 Institute on Programming Models is specifying, includes the specification of autonomously adaptable components. Despite some divergences (for instance DYNACO allows adaptable components to decide — possibly collectively — their own goals while GCM assumes that QoS contracts are submitted to components), DYNACO can be used to implement GCM autonomic controllers. Therefore, GCM as well benefits from understanding the relation between DYNACO and grid schedulers. While providing the *virtual node* abstraction for resource management and deployment (i.e. the mapping between job components and adaptable components), GCM does not give any indication on how autonomic controllers can place GCM components under the authority of the system-wide scheduler. In addition, WP6 Institute on Resource Management and Scheduling has acknowledged the need for schedulers to revise resource allocations even when jobs are active. Therefore, experience gained with the program benefits to the upcoming works of the institute on grid scheduling architecture.

4.3 Future directions

The proposed architecture has been implemented in part as a KOALA runner; in part as FRACTAL components that are included within DYNACO-based applications. The latter components act as proxies to the host runner. The current implementation does not support fault-tolerance; and it prevents applications from issuing grow requests.

During the program, we have chosen that claiming and releasing resources is under the responsibility of the instance of DYNACO, i.e. of *application* code. The main advantage is that only resources that will be surely used by the application are claimed, without needing the runner (or the scheduler) to implement an exhaustive list of relevant constraints applications may have on resource usage. However, the scheduler cannot authoritatively release some resources without minimal trust in application code. This shortcoming has still to be solved such that runners control resource management operations.

Currently, only applications that contain a single adaptable component are supported. The scheduler transparently handles when several competing adaptable components. But the impact of adaptability on co-allocation is still an open question. First, DYNACO will have to be extended in order to handle adaptation coordination, for instance building on the experience we have previously

gained with ACEEL [11]. Second, we will have to study whether the scheduler should take care of allocating resources consistently to all of the components of each job; or whether it should propose a bunch of resources and let applications pick out what they can use. Pros and cons of those two solutions will have to be investigated.

Acknowledgements

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265). Some experiments presented in this paper were carried out using the Grid'5000 experimental testbed, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners (see `https://www.grid5000.fr`). Some experiments presented in this paper were carried out using the DAS-3 experimental testbed (see `http://www.cs.vu.nl/das3`).

Papers

The following technical paper has been written from the results of the program.

- [BSMLE07] Jérémy Buisson, Ozan Sonmez, Hashim Mohamed, Wouter Lammers, and Dick Epema. "Scheduling malleable applications in multicluster systems". CoreGRID technical report TR-0092. This paper describes and evaluates the algorithms and policies that have been summarized in section 3.4. It has been submitted to Cluster 2007.

The following two technical papers have been written during the program. While not resulting directly from the work done during the program, they present previous works that are related to the program.

- [BAP07] Jérémy Buisson, Françoise André, and Jean-Louis Pazat. "Supporting adaptable applications in grid resource management systems". This paper, briefly summarized in section 3.1, results from the preliminary work that have consisted in making DYNACO-based applications execute with grid infrastructures such as GLOBUS, which operates the DAS-3 testbed. It analyzes how the management of malleable jobs can be designed in grid infrastructures and resource managers such as GLOBUS. It has been submitted to Grid 2007.
- [ABBPP07] Françoise André, Hinde Lilia Bouziane, Jérémy Buisson, Jean-Louis Pazat, and Christian Pérez. "Towards dynamic adaptability support for the master-worker paradigm in component based applications". This paper reports an approach derived from DYNACO in order to make adaptable master-worker components. It has been accepted at the CoreGRID Symposium 2007.

References

[1] The Distributed ASCI Supercomputer (DAS). http://www.cs.vu.nl/das3.

[2] Marco Aldinucci, Françoise André, Jérémy Buisson, Sonia Campa, Massimo Coppola, Marco Danelutto, and Corrado Zoccolo. An abstract schema modelling adaptivity management. In Sergei Gorlatch and Marco Danelutto, editors, *Integrated Research in GRID Computing*. Springer, 2007. proceedings of the CoreGRID Integration Workshop 2005.

[3] Marco Aldinucci, Sonia Campa, Massimo Coppola, Marco Danelutto, Domenico Laforenza, Diego Puppin, Luca Scarponi, Marco Vanneschi, and Corrado Zoccolo. Components for high performance grid programming in the grid.it project. In *Workshop on Component Models and Systems for Grid Applications*, June 2004.

[4] Marco Aldinucci, Alessandro Petrocelli, Edoardo Pistoletti, Massimo Torquati, Marco Vanneschi, Luca Veraldi, and Corrado Zoccolo. Dynamic reconfiguration of grid-aware applications in assist. In José C. Cunha and Pedro D. Medeiros, editors, *Proceedings of the 11th International Euro-Par Conference*, volume 3648 of *Lecture Notes in Computer Science*, pages 771–781, Lisbon, Portugal, September 2005. Springer.

[5] Rob Armstrong, Dennis Gannon, Al Geist, Katarzyna Keahey, Scott Kohn, Lois McInnes, Steve Parker, and Brent Smolinski. Toward a common component architecture for high-performance scientific computing. In *The Eighth International Symposium on High Performance Distributed Computing*, pages 115–124. IEEE, August 1999.

[6] Laurent Baduel, Françoise Baude, and Denis Caromel. Objected-oriented SPMD. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid*, pages 824–831, Cardiff, UK, May 2005.

[7] Eric Bruneton, Thierry Coupaye, Matthieu Leclerq, Vivien Quéma, and Jean-Bernard Stefani. The FRACTAL component model and its support in java. *Software: Practice and Experience*, 36(11-12):1257–1284, August 2006.

[8] Anca Bucur and Dick Epema. The maximal utilization of processor co-allocation in multicluster sys tems. In *IPDPS '03: Proceedings of the 17th International Symposium on Para llel and Distributed Processing*, page 60.1, Washington, DC, USA, 2003. IEEE Computer Society.

[9] Jérémy Buisson, Françoise André, and Jean-Louis Pazat. A framework for dynamic adaptation of parallel components. In G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P.Tirado, and E. Zapata, editors, *ParCo 2005*, volume 33 of *John von Neumann Intitute for Computing*, pages 65–72, Málaga, Spain, September 2005.

[10] Jérémy Buisson, Françoise André, and Jean-Louis Pazat. Afpac: Enforcing consistency during the adaptation of a parallel component. *Scalable Computing: Practice and Experience*, 7(3):83–95, September 2006. electronic journal (http://www.scpe.org/).

[11] Djalel Chefrour and Françoise André. 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*, February 2003.

[12] Geoff Coulson, Paul Grace, Gordon Blair, Wei Cai, Chris Cooper, David Duce, Laurent Mathy, Wai Kit Yeung, Barry Porter, Musbah Sagar, and Wei Li. A component-based middleware framework for configurable and reconfigurable grid computing. *Concurrency and Computation: Practice and Experience*, 18(8):865–874, July 2006.

[13] Ian Foster and Carl Kesselman. Globus: a metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[14] James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steven Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Journal of Cluster Computing*, 5:237–246, 2002.

[15] Eduardo Huedo, Ruben Montero, and Ignacio Llorente. A framework for adaptive execution in grids. *Software: Practice and Experience*, 34(7):631–651, March 2004.

[16] Katarzyna Keahey and Dennis Gannon. PARDIS : a parallel approach to CORBA. In *Proceedings of the Sixth IEEE International Symposium on High Performance Distributed Computing*, pages 31–39. IEEE, August 1997.

[17] William Lee, Stephen McGough, and John Darlington. Performance evaluation of the GridSAM job submission and monitoring system. In *UK e-Science All Hands Meeting*, pages 915–922, Notthingham, UK, September 2005.

[18] Moreno Marzolla, Paolo Adreetto, Antonino-Stefano Borgia, Alvise Dorigo, Alessio Gianelle, Matteo Mordacchini, Massimo Sgaravatto, Luigi Zangrando, et al. CREAM: a simple grid-accessible job management system for local computational resources. In *Computing in High-Energy and Nuclear Physics*, Mumbai, India, February 2006.

[19] Hashim Mohamed and Dick Epema. An evaluation of the close-to-files processor and data co-all ocation policy in multiclusters. In *2004 IEEE International Conference on Cluster Computing*, pages 287–298. IEEE Society Press, 2004.

[20] Hashim Mohamed and Dick Epema. The design and implementation of the koala co-allocating grid scheduler. In Peter Sloot, Alfons Hoekstra, Thierry Priol, Alexander Reinefeld, and Marian Bubak, editors, *Advances in Grid Computing - EGC 2005 (European Grid Conference, Amsterdam, The Netherlands, February 14-16, 2005, Revised Selected Papers)*, volume 3470 of *Lecture Notes in Computer Science*, pages 640–650, Amsterdam, February 2005. Springer.

[21] OGSA-BES Working Group. Ogsa basic execution services. Technical report, Open Grid Forum, 2007. Draft 33.

[22] CoreGRID Institute on Programming Model. Proposals for a grid component model. deliverable D.PM.02, CoreGRID, February 2006.

[23] Christian Pérez, Thierry Priol, and André Ribes. A parallel corba component model for numerical code coupling. In Manish Parashar, editor, *Proc. 3rd International Workshop on Grid Computing*, number 2536 in Lect. Notes in Comp. Science, pages 88–99, Baltimore, Maryland, USA, November 2002. Springer-Verlag. Held in conjunction with SuperComputing 2002 (SC '02).

[24] Maria-Teresa Segarra and Françoise André. A framework for dynamic adaptation in wireless environments. In *Technology of Object-Oriented Languages and Systems (TOOLS 33)*, pages 336–347. IEEE, 2000.

[25] Ozan Sonmez, Hashim Mohamed, and Dick Epema. Communication-aware job placement policies for the koala grid schedule r. In *E-SCIENCE '06: Proceedings of the Second IEEE International Confer ence on e-Science and Grid Computing*, page 79, Washington, DC, USA, 2006. IEEE Computer Society.

[26] Sathish Vadhiyar and Jack Dongarra. Self adaptability in grid computing. *Concurrency and Computation: Practice and Experience*, 17(2-4):235–257, February 2005.