

Optimization of Application Execution in the GridSpace Environment

Maciej Malawski, Joanna Kocot, Iwona Ryszka, Marian Bubak

malawski@agh.edu.pl

Institute of Computer Science and ACC CYFRONET

AGH University of Science and Technology

Al. Mickiewicza 30, 30-059 Kraków, Poland

Marek Wieczorek, Thomas Fahringer

marek@dps.uibk.ac.at

Institute for Computer Science, University of Innsbruck

Technikerstraße 21a, A-6020 Innsbruck, Austria



CoreGRID Technical Report
Number TR-0131

March 14, 2008

Institute on Resource Management and Scheduling

CoreGRID - Network of Excellence

URL: <http://www.coregrid.net>

Optimization of Application Execution in the GridSpace Environment

Maciej Malawski, Joanna Kocot, Iwona Ryszka, Marian Bubak
malawski@agh.edu.pl
Institute of Computer Science and ACC CYFRONET
AGH University of Science and Technology
Al. Mickiewicza 30, 30-059 Kraków, Poland

Marek Wieczorek, Thomas Fahringer
marek@dps.uibk.ac.at
Institute for Computer Science, University of Innsbruck
Technikerstraße 21a, A-6020 Innsbruck, Austria

CoreGRID TR-0131

March 14, 2008

Abstract

This paper describes an approach to optimization of execution of applications in the GridSpace environment. In this environment operations are invoked on special objects which reside on Grid resources what requires a specific approach to optimization of execution. This approach is implemented in the GridSpace Application Optimizer (GrAppO) which meets requirements imposed by the Grid environment as well as those specific to GridSpace model: dynamic nature of the environment, distributed sources of information, difficulty in defining suitable optimization criteria. The proposed solution applies three modes of optimization: short-, medium- and far-sighted one. The far-sighted optimization may be implemented with techniques known from the workflow scheduling research area, and for this purpose we have analyzed the possibility of using the ASKALON environment. We show that GrAppO enables to increase the quality of the GridSpace runtime performance by providing it with the most suitable objects to invoke operations on.

1 Introduction

Despite of many Grid scheduling and optimization algorithms and technologies, research in the area of new grid programming models leads to creation of novel execution environments for which standard optimization and scheduling techniques cannot be directly applied. In this paper, we focus on the optimization of application execution for a new high-level grid programming model and environment, called GridSpace [8].

GridSpace is a platform for collaborative application composition and execution, where developers can build the application with various technologies such as Web services, distributed components (MOCCA [11]) or legacy software accessible using grid systems such as EGEE [6]. These technologies provide different invocation semantics, e.g. WS provide stateless interactions, while stateful instances of MOCCA components can be dynamically deployed on the component container (called H2O kernel). The core idea behind GridSpace is to use a *scripting notation* for specifying application logic on a high level of abstraction, hiding the complexity of the underlying infrastructure from the programmer and leaving such decisions as resource discovery and selection to the runtime environment.

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

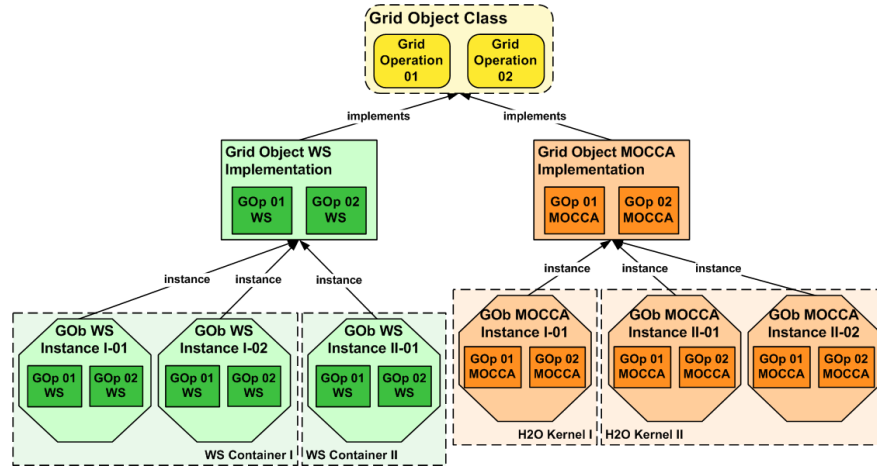


Figure 1: Grid objects, implementations and instances hierarchy.

In GridSpace, Ruby scripts are used to program the application, and, in addition to the standard Ruby library, special types of objects are introduced to represent activities performed on the grid (see Fig. 1). *Grid Operation* (*GOp*), which corresponds to an abstract method in terms of Object-Oriented programming, binds abstract methods, offering the same functionality. *Grid Object Class* (*GOB*) is an abstract class, or an interface, which declares a set of Grid Operations, and it is an abstraction of the same general functionality offered by different implementations. *Grid Object Implementation* implements functionality of its Grid Object Class; it is only a static entity which has to be instantiated (deployed into a resource) to allow invoking of its operations. A *Grid Object Instance* is an instance of a certain Grid Object Class. The instances which use the same implementation may differ only in resource they are deployed on. *Grid Resource* is able to host a Grid Object Instance.

The source code of the script provides only the information on Grid Object Classes – the selection of the actual instance is left to the runtime system. The choice of the instance, taking into account the structure of relations between the entities (see Fig. 1) is not trivial and must be performed by taking the following decisions: *which* Grid Object Implementation will be the most suitable to perform the processing; *which* existing Grid Object Instance of this Grid Object Implementation will be the most suitable; *whether* the Grid Object Instance should be chosen or a new one is to be deployed; *where* (on which Grid Resource) a new Grid Object Instance should be created.

In traditional grid environments dedicated components – a *scheduler* and *resource broker* are provided to answer such questions. As in GridSpace we need much broader functionality, the terms *optimization* and *optimizer* are introduced.

2 Related work

The problem of scheduling jobs to resources (including machines, storage, bandwidth, etc.) in Grid environment has been studied since many years [9, 12], and, depending on the type of applications various heuristics were applied. In the case of parameter sweep applications, typical heuristics include Min-min, Max-min and Sufferage [4]. More complex applications can be represented as workflows, therefore a large effort has been undertaken to investigate the solutions for workflow scheduling [3, 5]. For workflow applications, DAG-based scheduling algorithms can be successfully applied, such as HEFT and its extensions [19, 21], DLS [18], and FCP [14]. Since many solutions and scheduling tools already exist [20], the challenge was in selection of an appropriate scheduling model, and in adaption of available tools to the constraints of a specific scheduling problem.

3 Approach to Execution Optimization in GridSpace

The optimization of application execution in GridSpace can be presented as a special case of the generic optimization process in Grid computing [15]. Notions related to the GridSpace environment can be mapped into the general Grid

terminology in the following way: a Grid Object Instance represents a *resource*, a Grid Operation – a *job*, a GridSpace script – a Grid *application*. It is worth to notice that there is no direct control of resources and the optimizer can only act as a broker; it cannot guarantee that the task it schedules will be executed on the selected resource and is not able to manage the tasks after they are submitted for execution. The access to resources is not exclusive as the information about resources gathered by the optimizer can be imprecise or out-of-date. Furthermore, the optimizer cannot guarantee that the performance of scheduled jobs will not be reduced by some tasks executed by the local scheduler.

The optimizer may be used in one of three modes with different dependencies between subsequent tasks taken into consideration:

- Short-sighted optimization mode which implements the basic optimizer functionality: choosing an optimal performance solution for one task at a time.
- Medium-sighted optimization mode where tasks can be submitted to the optimizer in groups and for each task a resource is chosen on the basis of previous choices within the group. The tasks are not reordered, nor arranged in queues, and a group of tasks is mapped onto a group of resources.
- Far-sighted optimization mode, which is a suboptimal solution where resources are chosen for all tasks of an application and tasks may be reordered.

The modes are selected as a configuration option of the runtime system at application execution.

Thanks to the component architecture of GridSpace, the optimizer will be able to interact with other components which are a part of the runtime and have access to the information gathered by components of the Middleware layer (see Fig. 2). The only consumer of optimization results is the Grid Operation Invoker [2]. It is responsible for creating Grid Object Instances or contacting existing ones, and invoking Grid Operations on them. The Invoker provides an identification of Grid Object Class it is going to use, and obtains a Grid Object Instance or all the data necessary to create it.

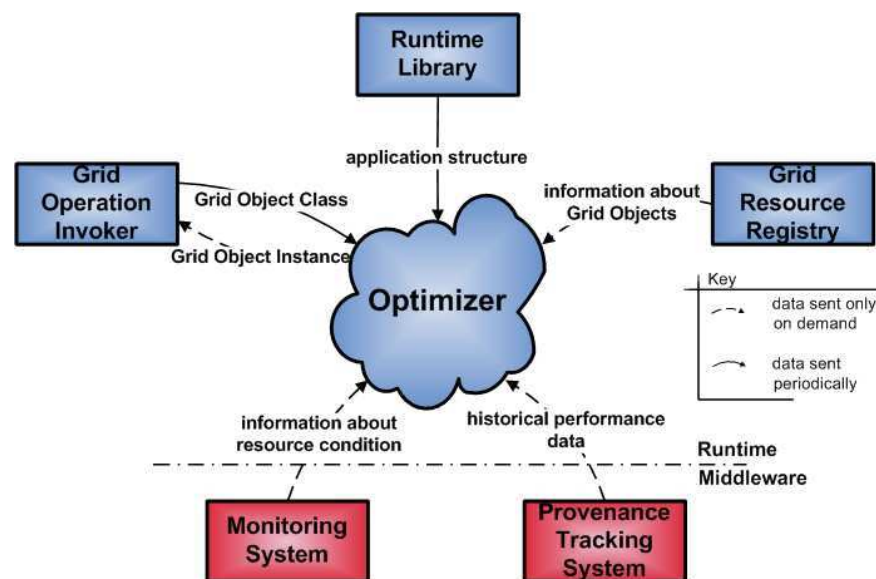


Figure 2: Optimizer placed in the context of neighboring components of GridSpace engine.

The process of optimization in GrAppO, in analogy to the schema introduced in [15] begins with the resource discovery phase. A list of all appropriate Grid Object Implementations including their running instances which are available to utilize at this moment for the Grid Object Class is generated. The only mandatory source of information is a Grid Resource Registry (GRR) which contains generic and static data, related to Grid Objects.

The system selection phase results in finding the best combination of a resource and a given job, i.e. determining the best Grid Object Instance on which the given Grid Object Operation could be performed, according to a specified metric. The phase involves also gathering dynamic information about resources from the Monitoring System. A

supporting component is the Provenance Tracking System – PROToS [1], which can be a source of data related to Grid Object Instances performance in their previous invocations. Additionally, for the far-sighted optimization a data about the structure of executed application (an *application graph*) is required. The source of such information is the core Runtime Library.

4 Far-sighted scheduling with ASKALON

Far-sighted optimization of GridSpace workflows can bring clear benefits in terms of execution time and other scheduling criteria, as dependencies between jobs may have a major influence on the selection of the most appropriate Grid Object Instances and Grid Resources. Implementation of a proper far-sighted optimization approach is a challenge for the GridSpace environment and its rich programming model based on scripting notation. Classical DAG scheduling heuristics like HEFT would not work well in such a model, and some additional optimization techniques would need to be developed in order to take advantage of all the provenance information (e.g., of the execution traces) provided by the supporting tools (PROToS). A promising approach may be to use the ASKALON Grid execution environment [7].

ASKALON was developed to compose and execute workflow applications on the Grid. ASKALON uses a high-level workflow description language called AGWL (Abstract Grid Workflow Language) and hides all the low-level implementation details of jobs from the workflow developer by storing them in the application repository called *Glare* [17] which is a part of the resource management service called *GridARM* [16]. The ASKALON *scheduler* applies different *full-graph* scheduling heuristics (e.g., HEFT or a bi-criteria scheduling algorithm called DCA) which are carried out based on the principle of *workflow conversion* [10]. In the process of workflow conversion, sophisticated workflows are dynamically transformed to simple DAGs, by evaluating the loops and conditionals present in the workflows. The latest version of ASKALON [13] is based on a more dynamic workflow processing model where scheduling is done only for the jobs which are ready for execution, rather than for a full graph.

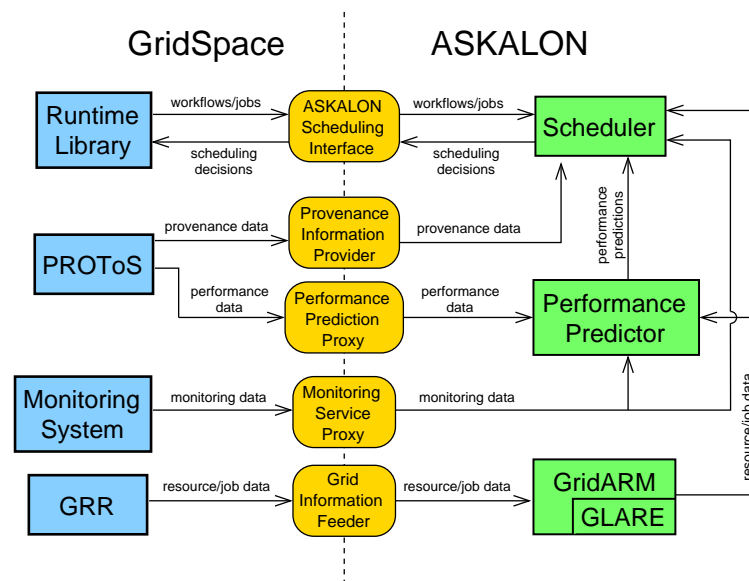


Figure 3: Integration between GridSpace and ASKALON

The feasibility of combining GridSpace and ASKALON depends on three major factors: (1) compatibility of the application representations; (2) compatibility of the resource/job representations; (3) reusability of the components supporting optimization in GridSpace (GRR, PROToS, and Monitoring System). Some preliminary feasibility studies performed by us showed that the set of workflow constructs used in AGWL is a superset of the set of constructs that are used or are planned to be used in GridSpace. The resource management model implemented in GridARM is also sufficient to represent all the resource information required in the GridSpace environment. On the other hand, the job representation model applied in Glare is simpler than the one in GridSpace, since is not based on the object-oriented paradigm, and allows for only two levels of abstraction: *activity type* (corresponding to a Grid Operation) and *activity*

deployment (corresponding to a method of a Grid Object Instance). However, this sparse number of abstractions is fairly enough to apply a successful workflow scheduling, as the abstractions of Grid Object Classes/ Implementations are designed for application development rather than for scheduling and execution. Finally, ASKALON will have to contact the Monitoring System of GridSpace in order to get the information regarding the current execution progress, since the job execution will be performed in the GridSpace environment (by the Grid Operation Invoker).

The proposed integration design of GridSpace and ASKALON is depicted in Fig. 3. In this system design, ASKALON acts only as an optimizer, while the whole job execution is carried out by the GridSpace environment. The most important element responsible for system integration is the *ASKALON Scheduling Interface*, which plays the role of a communication channel between the Runtime Library of GridSpace and the scheduler of ASKALON.

At the beginning of execution, the Runtime Library sends to the scheduler the whole application graph which is obtained from the script; as the execution proceeds, the scheduler is notified about the ongoing execution progress (job completions/failures). On the other hand, the scheduler sends to the Runtime Library the scheduling decisions made for the jobs which are ready for execution. This workflow processing model has been described in detail in [13].

Grid Information Feeder feeds the data from GRR into GridARM, which can be subsequently used by the scheduler. *Monitoring Service Proxy* is a simple proxy service which links the Monitoring System of GridSpace with the scheduler and Performance Predictor of ASKALON. Finally, the *Performance Prediction Proxy* provides the performance data from PROToS to the Performance Predictor, and *Provenance Information Provider* provides the provenance data to the scheduler.

5 Prototype Implementation and Tests

The optimizer prototype called GrAppO was implemented and integrated with the GridSpace engine with short- and medium-sighted optimization.

Testing involved measurement of the optimization quality from different points of view including unit tests, integration tests with other components of GridSpace, and finally quality tests to verify the usefulness of GrAppO operations in a given environment. Since the connections to Monitoring System and PROToS were not yet available, the tests were performed using mock components. As the objective function for the test the minimization of *makespan* was selected, which is a measure of the throughput of a heterogeneous computing system.

During the execution of the tests random data from external services (GRR, Monitoring System and PROToS) were generated. The data was complete, i.e. we assumed that no information is missing. We applied the constraints that the maximum expected execution time of Grid Operation is 500 seconds and the maximum time waiting for the resource to become available is set to 1500 seconds. The data generated in GRR had the following constraints: (1) Maximum number of Grid Object Classes is set to 100; (2) Maximum number of Grid Object Implementations per one Grid Object Class is set to 10; (3) Maximum number of Grid Object Instances per one Grid Object Implementation is set to 10; (4) Maximum number of resources per one Grid Object Class is set to 7.

Number of resources	Improvement of makespan	Percentage of improved results	Percentage of same results
5	4.03%	61.7%	4.0%
10	5.90%	61.1%	19.6%
20	5.47%	51.0%	45.8%
50	2.00%	20.2%	79.1%

Table 1: Improvement of makespan when comparing medium-sighted to the short-sighted optimization mode for 20 Grid Object Classes

The variable parameter in the test is a number of available resources and the number of Grid Object Classes for the optimization. For each given combination a list with names of Grid Object Classes to be optimized was randomly generated and passed to GrAppO. For each list the short-sighted optimization and the medium-sighted optimization were performed. For a given combination of number of resources and number of Grid Object Classes every test was performed 1000 times. The tests were performed for the cases that use the list of Grid Object Classes containing 10, 20 and 50 elements. The results of the tests for 20 Grid Object Classes are collected in Table 1. The meaning of the values in columns of the table is as follows: *Improvement of makespan* – presents the average improvement

of the makespan for all results while using the medium-sighted optimization in comparison to the short-sighted one. *Percentage of improved results* – shows the percentage of jobs that obtained a better result (makespan). *Percentage of the same results* – presents the percentage of jobs for which both optimization modes provided the same solution.

The improvement of the makespan depends on the number of available resources and the number of Grid Object Classes that are requested for the optimization. If the number of resources is significantly greater than the number of the objects for the optimization, the improvement of the makespan is the smallest. It is caused by the fact that with random generated data available Grid Object Instances or Grid Object Implementations are distributed on random locations and the probability of mapping two or more Grid Object Classes to the same resource is smaller than in the other cases. Therefore, the medium-sighted optimization suggests the solution very similar to the solution of short-sighted optimization.

The best improvement is observed for the situation when the number of available resources is smaller than the number of Grid Object Classes for the optimization. The rate of the improvement varies between 5.5% and 6.0%. With the increase of proportion of the number of Grid Object Classes to the number of available resources, the amount of improved results also increases. At the same time, the amount of the same results from both modes decreases.

6 Conclusions and future work

In this paper we have shown the analysis of the optimization problem in the case of GridSpace programming and execution environment. The proposed solution applies short-, medium- and far-sighted modes of optimization. Whereas the two former modes are directly feasible to implement, the far-sighted optimization requires applying more advanced techniques known from the workflow scheduling research area. For that purpose we have analyzed the possibility of using ASKALON, which offers dynamic workflow scheduling capabilities based on two alternative workflow processing models. The conclusion is that due to the compatibility of workflow models and the component-based design of both systems the integration of them will be feasible and may lead to further interesting research in scheduling of dynamic Grid applications.

The tests of two optimization modes currently available in GrAppO showed that with a suitable rate of available resources to Grid Object Classes the medium-sighted optimization mode can achieve the improvement at the level of 6% in comparison to the short-sighted mode. However, for these heuristics additional data from external components are necessary.

The future work will focus on implementation of the modules bridging GridSpace and ASKALON (Fig. 3), a more detailed feasibility study of the far-sighted mode, and extension of the Runtime Library with the capability of generation of an application graph required for far-sighted optimization. Integration with the Provenance and the Monitoring systems should result in some interesting research opportunities like investigation of different rescheduling techniques. We also plan conducting the performance tests on the real application scripts.

7 Acknowledgement

The authors express their gratitude to the GridSpace team, including Tomasz Gubała, Bartosz Baliś, Tomek Bartyński, Włodek Funika, Eryk Ciepiela, Daniel Haręźlak, Piotr Nowakowski, Darek Król and Kuba Wach. Special thanks go to the anonymous reviewers, for exceptionally detailed and helpful comments.

References

- [1] B. Baliś, M. Bubak, and J. Wach. Provenance Tracking in the ViroLab Virtual Laboratory. In *Proc. PPAM 2007, Seventh International Conference on Parallel Processing and Applied Mathematics*, LNCS, Gdansk, Poland, Sept. 2007. Springer. In print.
- [2] T. Bartyński, M. Malawski, T. Gubała, and M. Bubak. Universal grid client: Grid operation invoker. In *Proc. PPAM 2007, Seventh International Conference on Parallel Processing and Applied Mathematics*, LNCS, Gdansk, Poland, Sept. 2007. Springer. In print.
- [3] Berman, F. et al. New Grid Scheduling and Rescheduling Methods in the GrADS Project. *International Journal of Parallel Programming*, 33:209–229(21), June 2005.

- [4] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *Proceedings of 9th Heterogeneous Computing Workshop (HCW)*, pages 349–363, Cancun, Mexico, May 2000.
- [5] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbre, R. Cavanaugh, and S. Koranda. Mapping abstract complex workflows onto grid environments. *J. Grid Comput.*, 1(1):25–39, 2003.
- [6] EGEE Project. Website, 2006. <http://public.eu-egee.org/>.
- [7] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wiczorek. ASKALON: A Grid Application Development and Computing Environment. In *6th International Workshop on Grid Computing (Grid 2005)*, Seattle, USA, Nov. 2005. IEEE Computer Society Press.
- [8] T. Gubala and M. Bubak. Gridspace - semantic programming environment for the grid. In R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Wasniewski, editors, *PPAM*, volume 3911 of *Lecture Notes in Computer Science*, pages 172–179. Springer, 2005.
- [9] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. In R. Buyya and M. Baker, editors, *Grid Computing - GRID 2000, First IEEE/ACM International Workshop, Bangalore, India, December 17, 2000, Proceedings*, volume 1971 of *Lecture Notes in Computer Science*, pages 191–202. Springer, 2000.
- [10] M. Mair, J. Qin, M. Wiczorek, and T. Fahringer. Workflow Conversion and Processing in the ASKALON Grid Environment. *2nd Austrian Grid Symposium*, 2007.
- [11] M. Malawski, M. Bubak, M. Placek, D. Kurzyniec, and V. Sunderam. Experiments with distributed component computing across grid boundaries. In *Proceedings of the HPC-GECO/CompFrame workshop in conjunction with HPDC 2006*, Paris, France, 2006.
- [12] J. Nabrzyski, J. M. Schopf, and J. Weglarz, editors. *Grid Resource Management. State of the Art and Future Trends*. Kluwer Academic Publishers, 2003.
- [13] J. Qin, M. Wiczorek, K. Plankensteiner, and T. Fahringer. Towards a Light-weight Workflow Engine in the ASKALON Grid Environment. In *Proceedings of the CoreGRID Symposium*, Rennes, France, August 2007. Springer-Verlag.
- [14] A. Radulescu and A. J. C. van Gemund. On the complexity of list scheduling algorithms for distributed-memory systems. In *International Conference on Supercomputing*, pages 68–75, 1999.
- [15] J. M. Schopf. *Grid Resource Management. State of the Art and Future Trends*, chapter Ten Actions When Grid Scheduling. Kluwer Academic Publishers, 2003.
- [16] M. Siddiqui and T. Fahringer. GridARM: Askalon’s Grid Resource Management System. In *Advances in Grid Computing - EGC 2005 - Revised Selected Papers*, volume 3470 of *Lecture Notes in Computer Science*, pages 122–131, Amsterdam, Netherlands, June 2005. Springer Verlag GmbH, ISBN 3-540-26918-5.
- [17] M. Siddiqui, A. Villazon, J. Hofer, and T. Fahringer. GLARE: A Grid Activity Registration, Deployment and Provisioning Framework. In *SC ’05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 52, Washington, DC, USA, 2005. IEEE Computer Society.
- [18] G. C. Sih and E. A. Lee. A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures. *IEEE Trans. Parallel Distrib. Syst.*, 4(2):175–187, 1993.
- [19] H. Topcuoglu, S. Hariri, and M. you Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3):260–274, 2002.
- [20] M. Wiczorek, R. Prodan, and T. Fahringer. Scheduling of Scientific Workflows in the ASKALON Grid Environment. *SIGMOD Record*, 35(3), 2005.
- [21] Z. Yu and W. Shi. An Adaptive Rescheduling Strategy for Grid Workflow Applications. In *Proceedings of the 21st IPDPS 2007*, Long Beach, USA, Mar 26 -30 2007. IEEE Computer Society Press.