User-Transparent Scheduling for Software Components on the Grid

C. Dumitrescu, J. Dünnweber and S. Gorlatch
Department of Mathematics and Computer Science
The University of Münster
e-mail: dumitres, duennweb, gorlatch@uni-muenster.de

D.H.J. Epema
Electrical Eng., Mathematics and Computer Science Department
Delft University of Technology
e-mail: D.H.J.Epema@ewi.tudelft.nl

CoreGRID Technical Report
Number TR-0086
May 11, 2007

Institute on Programming Model &
Institute on Resource Management and Scheduling

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

User-Transparent Scheduling for Software Components on the Grid

C. Dumitrescu, J. Dünnweber and S. Gorlatch
Department of Mathematics and Computer Science
The University of Münster
e-mail: dumitres, duennweb, gorlatch@uni-muenster.de

D.H.J. Epema
Electrical Eng., Mathematics and Computer Science Department
Delft University of Technology
e-mail: D.H.J.Epema@ewi.tudelft.nl

CoreGRID TR-0086

May 11, 2007

Abstract

Grid applications are increasingly being developed as workflows using well-structured, reusable components. We argue that components with well-defined semantics facilitate an efficient scheduling on the Grid. We have previously developed a user-transparent scheduling approach for Higher-Order Components (HOCs) – parallel implementations of typical programming patterns, accessible and customizable via Web services. Our approach combines three scheduling techniques: using cost functions for reducing communication overhead, reusability of schedules for similar workflows, and the aggregated submission of jobs. We analyze the user-transparent scheduling from four perspectives, namely: the easiness of integration within already existing Grid scheduling systems, the gains for individual users, the resource provider advantages, and the robustness with respect to execution failures. We perform our evaluation using the KOALA Grid scheduler extended to support our user-transparent scheduling, which we run on the DAS-2 system combining over 200 nodes at five sites in the Netherlands. The experimental results show an increase in throughput by more than 100%, a descreasing of the response time by 50%, and a failure reduction by 45% for the considered scenarios.

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

1 Introduction

Grid technology provides a means for harnessing the computational and storage power of widely distributed collections of computers. Scheduling in a Grid environment is a complicated task and usually requires specific knowledge about the application being scheduled [3], in particular, the volume and the frequency of communication. Since the communication behavior of an arbitrary application cannot be foreseen during the setup of an application, it is typically the task of the application developer or the end user to provide information about the application's communication properties [5].

Grid applications are difficult to be developed from scratch: because of their complexity and scale, they increasingly rely on pre-packaged pieces of software which are called components, modules, templates, etc. in different approaches. In this paper we use Higher-Order Components (HOCs [16]) - reusable components that are customizable for particular applications using parameters which may be either data or code. HOCs include the required configuration to run on top of a standard Grid middleware [17] and can be remotely accessed via Web Services. Thereby, HOCs abstract over the technical features of Grid platforms and allow their users to concentrate on their applications. While providing more structure and reuse in the application development process, program components imply a change of focus for scheduling: it becomes mandatory to explore new solutions for mapping both single components and their compositions to available Grid resources.

We aim at user-transparent scheduling, i.e. techniques that free the end user from specifying application's communication behavior. Our approach to user-transparent scheduling makes use of: software components for building applications, scheduling cost-functions for lowering the communication costs, a reusable workflow technique for avoiding the need for a repeated scheduling when a workflow recurs, and an aggregated submission technique for avoiding multiple submissions of job to a single execution site. In this paper we present and analyze these techniques and their combined use for component-based Grid applications.

We use the KOALA system [21] as the basic scheduler for testing our user-transparent approach. KOALA supports co-allocation – the scheduling of parallel application on Grid resources. We enhance KOALA to schedule component-based applications using their specific communication properties.

The structure of this paper is as follows. First, we describe Higher-Order Components and the user-transparent scheduling approach; second, we introduce our integration and evaluation methodology; and third, we prove its advantages in the context of KOALA Grid scheduler and DAS-2, the Dutch Grid system, deployed at five sites in the Netherlands.

2 Context and Background

In this section, we describe the general context in which our approach to the user-transparent scheduling of component-based applications on Grids is developed. We first introduce the targeted environment of this work, and the concept of HOC – Higher-Order Component – for Grid application programming.

2.1 Environment Description

The main elements of the Grid environment considered in this work are as follows (see Figure 1):

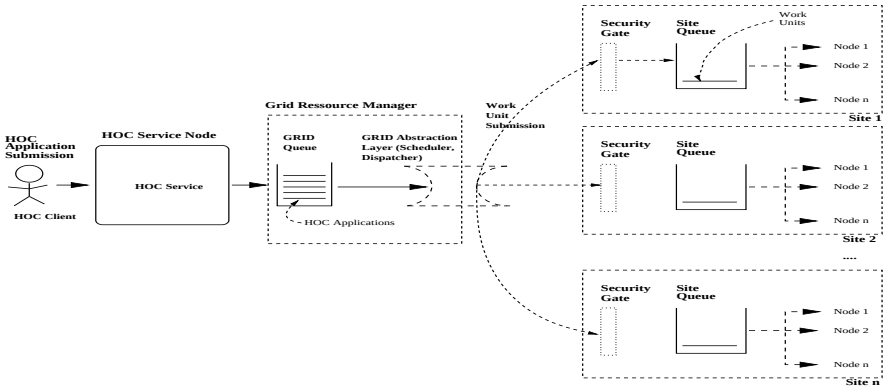

Figure 1: Grid Environment: High-Level Overview

- Node: a resource for computing and storing data;
- Site: a collection of nodes placed in a single administrative domain;
- Work unit (or job): a sequential code executed on a single node;
- Application: computation composed of work units;
- Resource Manager (RM): a specific software that allocates resources and monitors applications submitted by the users at a site level;
- Security Gate: a software that authenticates and authorizes user requests and invokes the RM whenever an application is submitted;
- HOC Client: a computer used for submitting HOC-based applications to the Grid built out of several sites;
- HOC Service Node: a resource providing a Web service for accessing a HOC implementation;
- Grid Resource Dispatcher: software that maps HOC applications onto a Grid, i. e. , it aggregates resources (nodes and sites) and reserves them for work unit execution.

In this environment, the HOC client submits a request for a specific HOC-based application by means of the HOC Web service. The HOC Web service generates a scheduling description of the application and passes it to the scheduler for running the application on the execution environment. Once the resource dispatcher aggregated the required amount of resources for the application, it returns a handle to the HOC Web service and steps aside from further interaction. Further, it is the task of the HOC Web service to actually submit the application to the Grid and to monitor its progress. In this manner, any HOC-based application can be scheduled over a Grid environment without the need of direct user interaction.

2.2 Higher-Order Components

HOCs are high-level programming constructs, pre-packaged with (parallel) implementations and the required middleware configuration files. Each HOC implements a generic pattern of parallel behavior with a specific communication structure. A HOC can be customized for a particular application by providing it with arguments which may be either data or application-specific code. HOCs were given their name (Higher-Order Components) because of their code parameters, in analogy with higher-order functions which accept functions as arguments. HOCs are made accessible to the Grid user via specialized Web services. Every HOC is by default configured to run on top of a specific middleware (Globus).

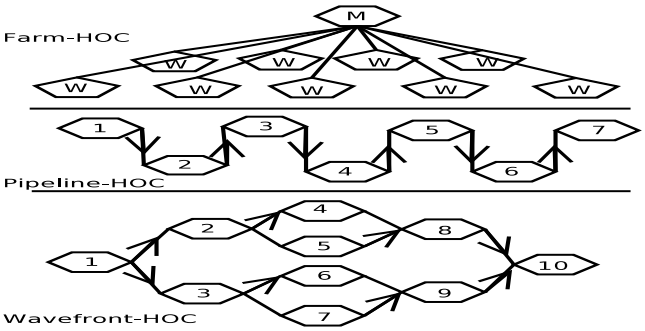

Figure 2: Examples of Communication Structures in HOCs

Currently three different HOCs are available: Farm-HOC [16], Pipeline-HOC [13] and Wavefront-HOC [14] (see Fig. 2). The Farm-HOC is used for running ”embarrassingly parallel” applications without dependencies between tasks. All farm implementations have in common the existence of a *Master* process that partitions data; the parts are then processed in parallel using multiple *Worker* processes. In contrast, in the Pipeline-HOC, all inputs pass through each pipeline stage in the same order, while the parallelism is achieved by overlapping the processing of several input instances. The third component studied in the paper, the Wavefront-HOC, implies a set of computations advancing as a hyperplane in a multidimensional variable space.

2.3 User-Transparent Scheduling for HOCs

Applications using HOCs have well-defined requirements in terms of data distribution and communication schemata, i. e. , which data is sent, when, and to how many processors, is completely determined by the HOC type and does not depend on the particular application where that HOC is used. Many different applications can be built using the same HOCs or combinations of them, but our user-transparent scheduling of HOC-based applications works independently of application-specific code and data parameters.

The following techniques are combined together in our implementation:

- *Cost functions* take in account the different characteristics of the environment in which a HOC application is executed. Various costs functions are employed for achieving an adequate distribution of resources to the work units (the "where?" part of scheduling). In [8], we have identified bandwidth-aware cost function, which provides the highest communication time reduction for the most HOCs;
- *Reusability of schedules for similar workflows* avoids successive job submissions with similar communication patterns. Job submission operations are expensive in Grids [10]. In our approach, different applications using the same HOCs will be mapped onto the same resources;
- *Aggregated submission of work units* exploits the fact that in many cases work units receive similar parameters for execution (similarly to an MPI application). Our scheduling system will place multiple job execution requests to avoid the necessity for individual HOC work unit submissions.

While user-transparent scheduling of single HOCs is already a complex operation, compositions of HOCs also occur in practice and must be scheduled on the Grid. This can be the case, when scientific applications are divided into several subproblems of different sizes and complexities. We schedule applications that make use of multiple HOCs composed together by applying our submission technique recursively to each single HOC in the composition, processing the global HOC first, and then down to inner ones.

3 Implementation of the Scheduler

We describe in this section our approach for implementing user-transparent scheduling in general and the metrics used to evaluate the performance of our implementation in different scenarios.

3.1 Integration of Existing Systems

We have chosen a layered solution for supporting HOC scheduling, because it provides the advantage of easily incorporating already existing scheduling solutions, and because of its flexibility in addressing various classes of parallel applications. We were interested in a minimal integration effort while taking advantage of already existing infrastructures. Based on existing Grid schedulers (KOALA [21], Pegasus [6], GridBus [4]) or brokers (GRUBER [9]), we have chosen a three-layers scheduling architecture [7] represented in Figure 3, and consisting of:

1. *Translation layer* for mapping the user's selection of a component to an associated communication pattern and for using a description of this model in order to select and provide as input to the cost management. This layer is also responsible for identifying whenever the reusable technique could be invoked and reusable submissions can be performed;
2. *Mapping and observation layer* for tracking allocated resource status and work units' progress and for taking action whenever a failure occurs and additional resources are needed. Any already existing Grid scheduling infrastructure is incorporated by this layer in our approach;
3. *Resource management layer* for acquiring and aggregating adequate resources to fulfill user objectives (reservation). This layer implements the aggregated submission of work units as described before and it is usually represented by different supporting tools of a Grid scheduler (i.e, runners for KOALA or site-selectors for GRUBER).

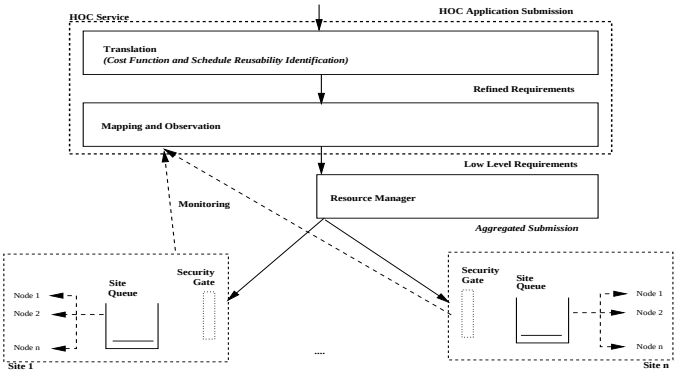

Figure 3: The Three-Layer, User-Transparent Scheduling Infrastructure

3.2 Performance Evaluation of Scheduling

We make two assumptions about the Grid on which our scheduling architecture operates. First, providers are interested in better utilizations of their resources (i.e., throughput); second, consumers want to achieve better performance for their applications and to acquire as many resources as possible when their applications require a lot of computational power (e.g., short response times). We employ two metrics for quantifying the performance of our user-transparent scheduling approach from both a user and a provider point of interest:

Metric-I. *Response Time* (RT):

$$RT = \sum_{i=1..N} RT_i/N \tag{1}$$

with RT_i being the individual job time response and N being the number of jobs processed during the execution period [11];

Metric-II. *Throughput* is defined as the number of work units executed on the Grid in a pre-defined interval of time [11].

There are many cases in which a Grid resource or service may fail in a dynamic, heterogeneous, and large-scale environment. Failures may occur in a Grid at infrastructure, middleware, application, and user levels, and may be transient or permanent; for a review of research focusing on causes of failures in Grid we refer to [18]. Due to the heterogeneity of Grids and their sheer size, failures appear much more often than in traditional parallel and distributed environments [10, 19]. We employ a third performance metric, namely:

Metric-III. *Failure Rate* [18], defined as the ratio of completed jobs to the total amount of jobs submitted from a fixed set of jobs.

4 KOALA-based User-Transparent Scheduling Evaluation

In this section, we report the results of our work to support user-transparent scheduling within the KOALA Grid scheduler and the experimental results for three HOC types on the Dutch Grid system, DAS-2 [15].

4.1 Integration Feasibility

For efficiently scheduling HOCs and applications built of them, we extended the KOALA scheduler with support for cost-based mapping of applications to resources, aggregated submission and reusable schedule identification. The standard KOALA version provided resource reservation and job co-allocation, basic job submission, tracking of job execution and re-submissions.

Our enhanced KOALA version adds the implementation of a specialized KOALA runner, the `MDRunner`, for handling HOC-based applications as explained in the following. Each HOC client starts such a `MDRunner`, while KOALA processes jobs as co-allocated jobs and performs adequate reservations.

In the resulting infrastructure, the responsibilities are distributed as follows:

- HOCs provide a high-level interface to the end user, allowing to compose applications in terms of patterns such as a wavefront or a pipeline;
- The `MDRunner` makes the scheduling of HOC applications transparent to the user. It automatically generates the scheduling requirements descriptions for instantiating a specific HOC pattern (Farm, Pipeline, Wavefront, etc); it also decides if aggregated submission or schedule reuse could be employed for the current application;
- The KOALA engine performs the resource acquisition and aggregation, by collecting information about resources, keeping track of application requirements and scheduling components to different Grid sites;
- The `MDRunner` and the KOALA engine share the functionalities of monitoring the application progress and the preservation of application-specific constraints, e. g. , time limits to be met.

The `MDRunner` implements six cost functions that optimize over different communication requirements [8]:

1. link count-aware: optimizes over the number of network connections a message has to pass;
2. bandwidth-aware: optimizes over the available bandwidth of a link;
3. latency-aware: optimizes over the latency of a link;
4. network utilization-aware: optimizes over the instantaneous network utilization at the submission time;
5. application communication-aware: optimizes over the communication pattern;
6. predicted variance-aware: optimizes over the predicted network bandwidth availability during the entire execution.

The gain in performance of each of these costs is analyzed in the following subsections.

4.2 Performance Results

We have performed 10 runs of each HOC type using synthetic applications that imitate the behavior of a real application in terms of communication patterns and computation requirements. We present our results for the three performance metrics as defined in the previous section. Each HOC-based application was composed of 15 to 20 work units and exchanged 20 messages with variable sizes between 1Mb to 10 Mb.

4.2.1 User Gains

We focus first on the response time for the user, achieved by our scheduling approach. The results are provided in Table 1, with all the values being expressed as percentages of the response time achieved under the KOALA's default scheduling policy. As can be observed, our techniques support a reduction of the response time by 60% in average.

Table 1: Average Response Time (%)

	Synthetic HOC Workload Type		
Cost Function	*Farm* 20×30×1M	*Pipeline* 20×30×10M	*Wavefront* 20×30×5M
CF (Close-to-File)	51.89	70.09	73.20
Link	45.37	55.06	65.40
Bandwidth	40.93	53.67	60.95
Latency	48.47	48.89	68.75
Network	48.54	50.52	67.15
Application	42.72	56.53	58.83
Predicted	44.02	53.63	57.96

4.2.2 Resource Provider Satisfaction

We show now the throughput performance when schedules are reused for HOC-based applications that exhibit the same workflow, i.e., they employ the same HOCs in identical order. Table 2 captures our results. We observe a high throughput improvement due to the schedule reuse: the reduction of the scheduling overhead allows to increase the total throughput by more than 100% in our test scenario, where synthetic HOC applications were submitted under a Poisson distribution. These applications involved 20 work units, 30 messages of sizes between 1 and 10 Mb (detailed in the Tables' headers) [8].

Table 2: Throughput Gains (%)

Cost Function	**Synthetic HOC Workload Type**		
	Farm 20×30×1M	*Pipeline* 20×30×10M	*Wavefront* 20×30×5M
CF	93.7	112.9	101.8
Link	100.7	116.3	161.0
Bandwidth	118.6	126.4	170.4
Latency	117.4	133.6	161.6
Network	106.3	134.8	170.2
Application	101.0	117.4	127.3
Predicted	118.2	123.6	194.0

4.2.3 Failure Analysis

Now, we turn our attention to analyzing the success rate of HOC applications on DAS-2. We classify failures into two types [10]:

- job failure is defined as the incapacity to run a job at a certain site, error that results in the entire workflow failure;
- workflow failure is defined as the incapacity to either start or complete the workflow; this error is usually caused either due to a failure in submission or in the resource acquisition mechanism.

For this analysis, we experimented with over 20k work units (corresponding to $\approx$ 1k workflows). Our results are presented in Table 3. While, in average, the rate of failures dropped under user-transparent scheduling approach, the work unit execution failures increased due to the DAS-2 uSLA [12] for stopping jobs longer than 15 minutes and our approach for schedule reuse. Thus, a more careful approach should be devised for these scenarios in the future.

Table 3: Detailed Failures (%)

Failure Type	*User-Transparent*	*Basic Scheduling*
Remote environment failure	5	11
GridFTP failure	0	0
Work unit execution failure	4	2
Workflow submission timeout failure	1	5
Total job failures	9	13
Total workflow failures	10	18

5 Related Work

In the context of application-to-resource dynamic binding, the optimal mapping problem has been approached in different ways for various types of systems.

For example, Aldinucci et al. [1] focus on the ASSIST coordination language - a language that allows to describe arbitrary graphs of modules which are connected by typed streams of data. A specialized compiler translates the graph

of modules into a network of processes on the Grid, under pre-specified rules. The difference in our approach is that we avoid application migration, because this feature is costly in practice, and we use instead the schedule reuse technique.

In [2], the Performance Evaluation Process Algebra (PEPA) for expressing performance models is described. PEPA [2] is also used in [1] for the analysis of the static information about a structured parallel application, given by its flow graph. In our work, we combine the information given by the static structure of a parallel program with monitoring information gathered at runtime.

Furmento et al. [20] analyze two main policies when considering application performance for Grids: *Minimum Execution Time* and *Minimum Cost*. They also introduce three cost models (unit cost per unit time and processor). Based on these assumptions, they derive various prediction results about the performance of a specific application (linear equation solver). Their initial results demonstrated the importance of finding the effective utilization of Grid resources by high performance applications, as well as the importance of information associated with various components in the system. The experiments were conducted in a small computational setting composed of at most three systems that had between 1 and 16 processors, while no Grids were considered.

6 Conclusions

In this paper we introduced the user-transparent scheduling approach for Higher-Order Components and analyzed it from four perspectives, namely: the easiness of integration within already existing Grid scheduling infrastructures, the gains for individual users, the resource provider satisfactions and the change in job failures. The motivations of this work are Grid applications that increasingly rely on workflows using well-structured, reusable components. Our approach combines several scheduling techniques, like cost-based technique for lowering the communication costs, reusable technique for avoiding the need for a repeated scheduling phase when a workflow recurs, and aggregated submission technique. We performed our evaluation by extending the KOALA Grid scheduler to support user-transparent scheduling, and by running the extended version on the DAS-2 testbed combining over 200 nodes at five sites in the Netherlands. Our experimental results show an increase in throughput with more than 100%, a lowering of the response time by 50%, and a failure reduction by 45% for the considered scenarios.

References

[1] A. Benoit and M. Aldinucci. Towards the Automatic Mapping of ASSIST Applications for the Grid. In *Proceedings of CoreGRID Integration Workshop*, University of Pisa, Italy, Nov. 2005.

[2] A. Benoit, M. Cole, S. Gilmore, and J. Hillston. Evaluating the performance of pipeline-structured parallel programs with skeletons and process algebra. *Parallel and Distributed Computing Practices, special issue on Practical Aspects of High-level Parallel Programming PAPP2004*, 2005.

[3] A. I. D. Bucur and D. H. J. Epema. The influence of communication on the performance of co-allocation. In *JSSPP '01: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 66–86, London, UK, 2001. Springer-Verlag.

[4] R. Buyya, D. Abramson, and J. Giddy. An economy driven resource management architecture for computational power grids. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, 2000.

[5] A. Dan, C. Dumitrescu, K. Ranganathan, and M. Ripeanu. A Layered Framework for Connecting Client Objectives and Resource Capabilities. *International Journal of Cooperative Communication Systems*, 2006. To appear.

[6] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny. Pegasus : Mapping scientific workflows onto the grid. In *2nd EUROPEAN ACROSS GRIDS CONFERENCE*, Nicosia, Cyprus, 2004.

[7] C. Dumitrescu, D. Epema, J. Dünnweber, and S. Gorlatch. User Transparent Scheduling of Structured Parallel Applications in Grid Environments. In *HPC-GECO/CompFrame Workshop held in Conjunction with HPDC'06*, 2006.

[8] C. Dumitrescu, D. H. Epema, J. Dünnweber, and S. Gorlatch. Reusable Cost-based Scheduling of Grid Workflows Operating on Higher-Order Components. Technical Report TR-0044, Institute on Resource Management and Scheduling, CoreGRID - Network of Excellence, 2006.

[9] C. Dumitrescu and I. Foster. GRUBER: A Grid Resource Usage SLA BrokER. In *Proc. of 11th International Euro-Par Conference (Euro-Par'05), Portugal*, 2005.

[10] C. Dumitrescu, I. Raicu, and I. Foster. Experiences in running workloads over Grid3. In *Grid and Cooperative Computing (GCC)*, 2005.

[11] C. Dumitrescu, I. Raicu, and I. Foster. DI-GRUBER: A Distributed Approach for Resource Brokering. In *Proc. of SuperComputing Conference, Seattle, USA*, 2006.

[12] C. Dumitrescu, M. Wilde, and I. Foster. A Model for Usage Policy-based Resource Allocation in Grids. In *Policies for Distributed Systems and Networks, 2005. Sixth IEEE International Workshop on Policy*, pages 191 – 200, June 2005.

[13] J. Dünnweber, S. Gorlatch, A. Benoit, and M. Cole. Integrating MPI-Skeletons with Web services. In *Proceedings of the International Conference on Parallel Computing, Malaga, Spain*, September 2005.

[14] J. Dünnweber, S. Gorlatch, S. Campa, M. Danelutto, and M. Aldinucci. Using code parameters for component adaptations. In S. Gorlatch, editor, *Proceedings of the CoreGRID Integration Workshop, Pisa, Italy*, November 2005.

[15] Dutch University Backbone. The distributed ASCI supercomputer 2 (DAS-2), 2006.

[16] S. Gorlatch and J. Dünnweber. From Grid Middleware to Grid Applications: Bridging the Gap with HOCs. In *Future Generation Grids*. Springer Verlag, 2005.

[17] M. Humphrey, G. Wasson, J. Gawor, J. Bester, S. Lang, I. Foster, S. Pickles, M. M. Keown, K. Jackson, J. Boverhof, M. Rodriguez, and S. Meder. State and events for Web services: A comparison of five WS-resource framework and WS-notification implementations. In *14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, 2005.

[18] A. Iosup, D. H. Epema, C. Franke, A. Papaspyrou, L. Schley, B. Song, and R. Yahyapour. On Grid performance evaluation using synthetic workloads. In *The 12th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, Saint Malo, FR, June 2006.

[19] A. Iosup and D. H. J. Epema. GrenchMark: A framework for analyzing, testing, and comparing grids. In *6th IEEE/ACM Int'l Symposium on Cluster Computing and the Grid (CCGrid)*, 2006.

[20] A. Mayer, S. McGough, and N. Furmento. ICENI: Optimisation of component applications within a grid environment. In *Parallel Computing Amsterdam*, 2002.

[21] H. Mohamed and D. Epema. The design and implementation of the KOALA co-allocating grid scheduler. In .-. LNCS 3470, editor, *Proceedings of the European Grid Conference, Amsterdam*, 2005.