

User-Transparent Scheduling of Structured Parallel Applications in Grid Environments

C. Dumitrescu and D.H.J. Epema

*Electrical Eng., Mathematics and Computer Science Department
Delft University of Technology*

e-mail: C.Dumitrescu, D.H.J.Epema@ewi.tudelft.nl

J. Dünnweber and S. Gorlatch

*Department of Mathematics and Computer Science
The University of Münster*

e-mail: duennweb, gorlatch@uni-muenster.de



CoreGRID Technical Report
Number TR-0034
March 9, 2006

Institute on Programming Model (WP 3) &
Institute on Resource Management and Scheduling (WP 6)

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

User-Transparent Scheduling of Structured Parallel Applications in Grid Environments

C. Dumitrescu and D.H.J. Epema

Electrical Eng., Mathematics and Computer Science Department
Delft University of Technology

e-mail: C.Dumitrescu, D.H.J.Epema@ewi.tudelft.nl

J. Dünnweber and S. Gorlatch

Department of Mathematics and Computer Science
The University of Münster

e-mail: duennweb, gorlatch@uni-muenster.de

CoreGRID TR-0034

March 9, 2006

Abstract

The problem of scheduling parallel applications on Grids is notoriously difficult: schedulers must consider the heterogeneity of involved resources and management systems, and they often require the users to provide information about the expected application behavior. We suggest that increasingly popular structured programming approaches, using components with well-defined semantics, facilitate a more efficient and user-transparent scheduling. We illustrate our approach to scheduling using HOCs (Higher-Order Components) that capture typical patterns of parallelism (Farm-HOC, Pipeline-HOC, etc.) and come as Web Services with pre-packaged implementation and middleware setup. We take a particular Grid scheduler, KOALA, and enhance it for Grid applications built of HOCs: we propose and implement a novel scheduling algorithm that optimizes communications in such structured applications. We report experimental results of running our scheduler on the DAS-2 Grid testbed combining over 200 nodes at five sites in the Netherlands.

1 Introduction

Grid technologies provide a means for harnessing the computational and storage power of widely distributed collections of computers. Grid computing has emerged as an important new field, distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation.

While Grid infrastructures have become almost commonplace, the performance and the automation of Grid resource management tools are far from ubiquitous because of the complexity of the workloads that occur in practice [21, 26, 8] and the impossibility of tight control over physical resources at the Grid level. Running applications efficiently in such environments is often a challenging problem due to the scale and heterogeneity of Grids [10, 14].

The fundamental scheduling operation of mapping application components to Grid resources is complicated and usually requires specific knowledge about the application being scheduled [3]. The frequency of communication and the amount of data being communicated matter strongly. Since the communication behavior of an arbitrary application cannot be foreseen during the setup of the runtime platform, it is typically the task of the application developer or end

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

user to provide information about the application’s performance model [7] and additionally to specify the application’s resource requirements.

Grid applications are rarely 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. While providing more structure and reuse in the application development process, such approaches require 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. One concept for the component-based development of Grid applications is given by the Higher-Order Components (HOCs) [15] programming model, which is an extension of the skeleton approach to parallel programming [6]. Skeletons are reusable implementations of parallel computing schemata, and HOCs make skeletons accessible in a Grid environment via Web Services [17] that abstract over the technical features of the particular runtime platform.

In this paper, we study the task of scheduling HOC-based Grid applications. HOCs play for us the role of an exemplary structured approach, as a step on the way towards our ultimate goal: to develop general scheduling techniques, which would be useful in a broader context of Grid applications based on reusable pieces of software. Thus, we explore the problem of optimally scheduling HOC-based applications in particular and any structured parallel application in general in large-scale distributed environments. In addition to efficiency, another aim of our scheduling approach is user-transparency: the user should be freed from the task of building a detailed performance model for the application to be scheduled.

The main contributions of this paper are, first, the identification of the key functionalities of a scheduling infrastructure adequate for the requirements of Grid applications built of reusable components; second, the integration of these functionalities and their implementation in the KOALA scheduling system [21]; third, a novel scheduling algorithm that provides an optimal mapping (under the considered communication cost function) for HOC-based applications to available Grid resources; and fourth, the simulation and experimental results that demonstrate the efficiency of our approach to the scheduling of structured parallel applications.

2 The Problem Context

In this section, we describe the general context in which our approach to the user-transparent scheduling of HOC-based applications [15] on Grids is developed. We first introduce the HOC concept for Grid application programming and then describe the KOALA Grid scheduler.

2.1 The HOC Class of Structured Parallel Applications

Higher-Order Components (HOCs) [15] provide Grid users with 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 both data and 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.

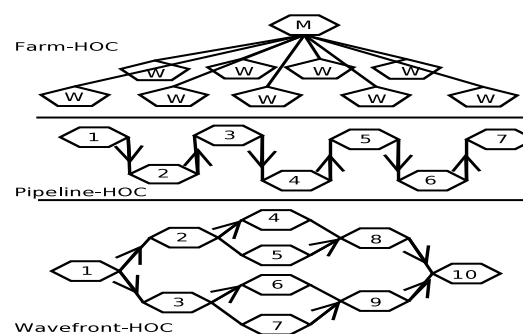


Figure 1: Examples of Communication Structures in HOCs

Currently there are three different HOCs available, Farm-HOC [15], Pipeline-HOC [11] and Wavefront-HOC [12] (see Fig. 1). The Farm-HOC is used for running ”embarrassingly parallel” applications without dependencies between

tasks. Even though there are several possibilities for implementing the farm pattern executed by the Farm-HOC, all 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 considered in this paper, the Wavefront-HOC, implies a set of computations advancing as a hyperplane in a multidimensional variable space, called diagonal sweep wavefront in the two-dimensional case.

Practically, every HOC has some inherent communication properties which are preserved for any application where this HOC is used. In the Pipeline-HOC, e. g., only successive stages communicate, while in the Farm-HOC, only the master communicates with all workers.

2.2 Grid Scheduling Infrastructure

The scheduling technologies used in this work are the *Sun Grid Engine* (SGE) [4] and KOALA [21]. KOALA has been designed and implemented by the PDS group in Delft in the context of the *Virtual Lab for e-Science* (VL-e) [16] project. The main feature of KOALA is its support for co-allocation; that is, the simultaneous allocation of resources in multiple clusters comprising a Grid to a single application consisting of multiple stages. Currently, KOALA supports processor and memory co-allocation and makes use of the SGE local resource managers.

In the KOALA infrastructure, every host used for submitting jobs has at least one KOALA *runner* tool in operation, while one centralized Grid host runs the KOALA *engine*. All runners communicate with the engine via a proprietary light-weight protocol based on exchanging two distinct types of messages (*requests* and *answers*). Applications are executed by submitting single-CPU jobs (e. g., a code parameter of a HOC, such as a pipeline stage) to the runners which handle the reservations of resources. The runners use Globus as the middleware platform and take as their input partially filled-in job descriptions in the Globus-typical *Resource Specification Language* (RSL) format, without the *resource manager contact string* specified [25]. This RSL-document is passed to the KOALA engine which selects the appropriate execution sites, based on previously gathered monitoring information (collected using Globus MDS). Once the KOALA engine has completed the RSL documents, they are sent back to the execution sites which trigger the execution of the application on the Grid (i. e., on the Grid nodes belonging to one site) using the *Globus Resource Allocation Manager* (GRAM) [25].

KOALA provides different kinds of runners, specialized for different types of applications:

- The `KRunner` is the most basic KOALA runner, used for executing standard GRAM jobs, i. e., programs directly executable upon UNIX.
- The `DRunner` is used for starting multiple jobs (managed using DUROC [24] in Globus) that communicate via MPI and require an appropriate runtime environment.
- The `GRunner` is used for scheduling Ibis jobs [27].

KOALA makes use of the *Close-to-File* policy (CF) or the *Incremental Claiming* policy (IC) for scheduling jobs. Under CF, execution nodes are chosen depending on the estimated time of transferring the input files to the nodes, while under IC, the execution nodes are selected at runtime [21].

Our contribution in this paper is a new runner for KOALA, which we call `MDRunner` and which extends the `DRunner` by a new communication and input file-aware strategy for scheduling HOC-based applications. HOC applications using the `MDRunner` benefit from enhanced data processing capabilities at the execution sites and from an advanced submission management, provided by the KOALA engine.

3 HOC Scheduling Requirements

The purpose of this section is to identify the key functionalities generally required by any scheduling infrastructure, and by KOALA in particular, for an automated mapping of structured parallel components to available Grid resources. Applications using structured components 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 and does not depend on the particular application where that HOC is used. In any application of the Farm-HOC, e. g., it is the master that sends all data.

By exploiting the information hidden in the components' parallelism structure, communication times among distributed components, as well as resource occupancy times can be decreased, thus minimizing the overall wall-clock execution time of applications. Structured programming components like HOCs help detecting the most busy communication links in an application easily. The decreasing of communication time is achieved by mapping busy links to better connected nodes whenever possible. Independently of the frequency of a communication, the sending of bulky data over distant connections should be avoided. These problems concern every single execution site manager (the KOALA runners).

3.1 Key Scheduling Functionalities for HOCs

Various requirements emerge when considering the scheduling of any application in large and distributed systems. The scheduler objectives are composed of synchronized resource reservations, bandwidth allocations and/or execution deadlines at various application stages. Thus, we are interested in identifying the lacking functionalities of the existing tools [21, 22] for user-transparent structured parallel application scheduling [8, 7]. Tonello et al. [26] identify a detailed list of core functions that need to be addressed when building a scheduling infrastructure. Some of these functionalities are: naming, search, monitoring, forecasting, reservation, negotiation, execution, translation, etc.

We argue that a Grid scheduler enabling the user-transparent scheduling of applications must fulfill at least the following four main functionalities [8, 7]:

- **Automated requirements translation** consists in identifying the performance model of a parallel application in an automated way. Sometimes, applications require complex configurations (CPUs, disk, network or higher level utilities), but we would like to free the end users from explicitly specifying these requirements. In the case of HOC-based applications, application requirements are directly extracted from the parallelism schema implemented by the type of HOC being used, which is known, once a HOC is selected and before data processing starts and the jobs must be submitted by the scheduler [8, 10, 7]. One such requirement, which emerges, e. g., in applications using the Pipeline-HOC with frequent communication among stages, is a minimum amount of MB/s bandwidth.
- **Mapping of application stages to resources** is the finding of the optimal resources for each application stage in order to both enable execution and achieve best performance. When the resources are clusters, one of the most popular examples for making such matches is the Condor resource manager [19]. In Grid environments, serving simultaneous requests that compete for the available resources is especially difficult. A Grid scheduler must handle the assignment of jobs to resources in a way that maximizes the delivered 'utility' (for example, throughput) and it must meet the resource providers' rules and objectives [8].
- **Resource reservation** is the step in which the actual physical resources are reserved prior to application execution. Because multiple resources are involved in complex requests, a request coordination mechanism is required for the resource acquisition [21]. This mechanism must take into account that resources may belong to multiple administrative domains [8, 23].
- **Agreement violation handling** addresses the challenging problem arising in large and distributed systems where resources may be shut down during a job execution, be improperly setup or simply fail to execute a job. This problem is addressed by providing multiple alternative policies which can be employed to fulfill user objectives in a transparent manner when violations occur.

After identifying the four functionalities above, let us focus on how they can be implemented. We continue by proposing a generic three layer scheduling infrastructure that addresses all of the four scheduling functionalities in a user-transparent manner. Next, we describe our implementation in the specific context of the KOALA Grid scheduler [21].

3.2 Scheduling Layers

One can imagine a monolithic Grid scheduler that provides the above four functionalities. However, we suggest that a layered solution is a better approach, because it provides the advantage of plugging in already existing solutions for one functionality or another, and also because of its flexibility in addressing various classes of parallel applications. In our concrete case, using the communication semantics of HOCs, we propose a three-layer scheduling infrastructure for

addressing the identified functionalities. We consider three to be the minimal number of layers in terms of addressing the above functionalities while also maintaining an adequate decoupling of them. The layers can be implemented using any existing scheduling infrastructure that addresses the resource management and monitoring.

We propose the following three layers (see Fig. 2):

- **Translation layer** is introduced due to our aim for transparency in this work. It maps the user’s selection of a component to an associated performance model and passes a description of this model, including the involved processes, the tasks being assigned to them and, potentially, dependencies among these tasks, further to the other scheduling layers. Thus, users only chose HOCs and are freed from describing performance models themselves.
- **Mapping and Observation layer** keeps track of the allocated resource status and application stages’ progress and takes action whenever an agreement violation occurs and additional resources are needed. Violation of user-specified agreements may lead to a re-mapping of application components, which is triggered in this layer.
- **Resource management layer**, based on the previous model identification, acquires and aggregates adequate resources to fulfill user objectives (reservation). Practically, whenever the scheduler has a choice, it will select processors that best match the application communication pattern, with respect to the processor interconnections.

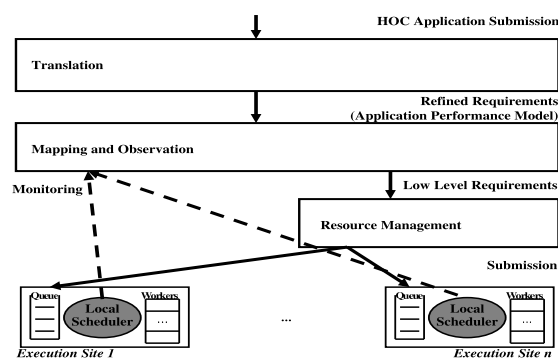


Figure 2: The Three-Layer User-Transparent Scheduling Infrastructure

We argue that our layered scheduling approach is not limited to HOCs only, but suitable for enabling user-transparent scheduling in any structured parallel application built upon software stages with well-defined semantics.

4 Illustrating the Key Functionalities for the Case of Scheduling HOCs by means of KOALA

In this section, we enhance the KOALA Grid scheduler. Two of the functionalities outlined in Section III. A are already addressed via KOALA’s parallel application and co-allocation facilities [21]: the mapping of application stages to resources and the resource reservation. The missing functionalities are the automated requirements translation and a HOC-aware scheduling policy for handling agreement violations. In order to illustrate these functionalities in practice and provide the reader with an intuitive understanding, we analyze a concrete case study for scheduling a HOC-based application.

4.1 HOCs Integration with the KOALA Grid Scheduler

As an example, we consider an image processing application using the MPI-based HOC [11]. In this example, the role of HOCs is to abstract over the Globus-managed resources [17] and the MPI processes running on these resources, such that the client application only has to issue a Web Service request to run the application.

Fig. 3 outlines the relation among clients, Web Services, HOCs, the KOALA engine and the MDRunner which we specifically developed for scheduling HOC-based applications. In the figure, *Client 1* contacts *Web Service 1* which makes use of the MPI-based Pipeline-HOC. In [11], it was shown that a gateway between the Web Service and the MPI-environment can be built by issuing an `mpi-run` command from inside a HOC. In the KOALA infrastructure, instead of `mpi-run`, the MDRunner is invoked. This allows the Pipeline-HOC to submit single stages separately and

have them scheduled dynamically, while in [11], the target nodes were statically assigned in a fixed part of the HOC configuration. *Client 2* runs a different application, based on the Wavefront-HOC [12], which does not use MPI. As can be seen in the figure, the MDRunner is also used for scheduling this application which is fully written in Java. A special feature of the MDRunner is that it can apply its scheduling policies not only to MPI-programs, but also to programs that would have used any of the more basic runners, such as the KRunner, if the MDRunner was not available. In Fig. 3 *Execution Site 1* runs stages of a pipeline on top of an MPI environment, which is managed by the Globus Dynamically-Updated Request Online Coallocator (DUROC) [24]. When only Java is used, as it is the case for the Wavefront-HOC, the MDRunner executes the application together with the required virtual machine on a site, which runs SGE [4] as its *Local Scheduler*.

The necessary scheduling arrangements consist in resource reservation, the mapping of HOCs to resources, the submission of jobs (i. e. , the transfer of the application data to be processed), the tracking of the execution and job re-submissions, in case of agreements violations.

To enable the scheduling of HOCs, we have developed the MDRunner (Modified DUROC Runner). The MDRunner internally maintains a list which holds a communication model for each HOC. From the KOALA engine, the runner gets the list of available sites and monitoring information about these sites. To this information, the MDRunner applies the BWCF algorithm described next. We envisage that in the future the algorithm may be migrated to the engine itself, such that the MDRunner only needs to download the list of sites to where the application is actually being dispatched.

In our architecture, the scheduling responsibilities are distributed as follows:

- HOCs provide a high-level interface to the clients, allowing to compose applications in terms of schemata such as a Wavefront-HOC or Pipeline-HOC, without exposing the user to any of the technical details related to running such schemata on the Grid.
- The MDRunner maps the type of HOC being used to a performance model from its internal list. For our example application from [11], this model requires the reservation of an MPI-environment, wherein each process runs one pipeline stage. The MDRunner translates these requirements into a lower-level job specification (in the Globus RSL format). When input, consisting solely in data, is submitted to a HOC via its Web Service interface, the MDRunner combines this data with the job specification. The result is a full job description in RSL, which is valid for using Globus GRAM to submit the described job to one of the remotely located execution sites.
- The KOALA engine performs the resource acquisition and aggregation functions; i. e. , the assignment of target nodes is handled by one centralized module, while distinct KOALA runners are provided for every HOC.
- The MDRunner and the KOALA engine share the functionality of monitoring the application progress and the preserving of application-specific constraints, e. g. , time limits to be met.

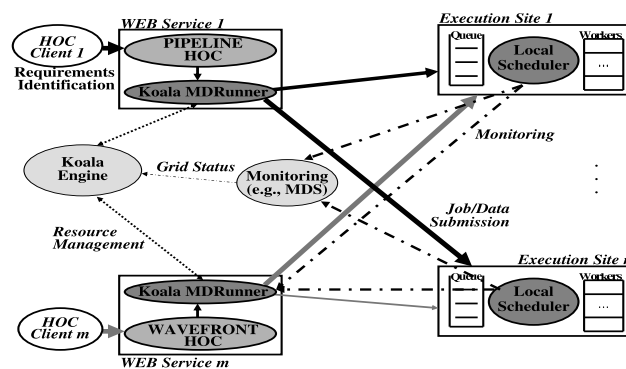


Figure 3: HOC-based Applications Integration within the KOALA Scheduler

4.2 A HOC-Aware Scheduling Strategy

The close-to-file policy (CF) [21], as implemented in KOALA, reduces the costs associated with transferring the input/output files needed for running an application. But CF does not take into account the communication requirements

implied by a distributed parallel application. One of the obstacles concerning a communication-aware scheduling strategy is the difficulty of predicting the communication behavior of an application before it is executed. In our implementation, the `MDRunner` uses the algorithm introduced in the following, which combines the close-to-file policy with communication cost awareness. Our algorithm, which we call BWCF (for BandWidth-aware and Close-to-File scheduling), is based on the KOALA CF [21], while it also incorporates a communication cost function. The main advantage of our BWCF algorithm, as compared to KOALA's greedy CF algorithm, is that we can provide an optimal mapping for applications with inter-process communication.

The cost function considered in this work weighs remote communications with three times the costs of a local communication (Worst Fit Policy) [18]:

$$Cost(j) = 1 * ICL_j + 3 * RCL_j \quad (1)$$

where ICL_j represents the number of intra-cluster links a component j has and RCL_j represents the number of inter-cluster links the same component j has. Under this cost policy, a site N_j is selected for executing the component j iff it provides the minimal communication cost between component j and all its peers. For identifying the overall application communication cost of an application, where several alternatives are possible at each step, we employed the shortest-path approach described in [5].

In the general case, the complexity increases with the number of communication links per stage. In the worst case (each stage exchanges messages with all the others, which implies $n-1$ links for each stage), the complexity of BWCF is $O(n^2 * N)$ due to line 15. The complexity of the BWCF algorithm depends though on the type of HOC being used and the number of available execution sites. For the Pipeline-HOC, it is $O(n * N)$ for a number of HOC instances n scheduled on a number of available sites N .

Algorithm BWCF ($J[n]$, $S[N]$)

/ n - the number of job stages; N - the number of sites */*

/ J[n] - the application stages; S[N] - the list of sites */*

1. order job stages according to their dependencies

/ Build the set of mappings for the 1st stage */*

2. S_1 = set of potential execution sites for the first stage

3. **if** ($S_1 \neq \emptyset$) **then**

4. **for each** ($E \in S_1$) **do**

5. estimate the file transfer cost TF_E

6. $Q_1.add(E, (-1, TF_E))$

7. **else**

8. **return** mapping failure (no available execution sites)

/ Build the set of mappings for the jth stage */*

9. **for each** (job stage j from 2 to n) **do**

10. S_j = set of potential execution sites

11. **if** ($S_j \neq \emptyset$) **then**

12. **for each** ($E \in S_j$) **do**

13. estimate file transfer $TF_{(E)}$ cost

14. **for each** ($F \in elements(Q_{j-1})$) **do**

/ Estimate communication cost */*

15. $TC_{(F,E)} = Cost(E, AL(Q_{j-1}(F)))$

16. **if** $TF_{(F,E)} + TC_{Input,E} + Q_{j-1}[F] < C$ **then**

17. $x = E$ and $F_x = F$

18. $C = TF_{(F,E)} + TC_{I,E} + Q_{j-1}[F]$

19. $Q_j.add(x, (F_x, C))$
 20. **else**
 21. **return** mapping failure (no available execution sites)
- /* Extract the final HOC application mapping */
22. **for** $i = n$ **downto** 1 **do**
 23. $N_i = E$ such that $MIN(Q_j[F])$ and $F = Q_i[E]$

4.3 Proving the Optimality of BWCF Algorithm

While the BWCF algorithm tries all possibilities in each step and selects the most appropriate one in terms of the cost function, it remains to prove the optimality of the obtained solution. We conduct the proof by contradiction. Let us assume that a better solution exists. Such a solution must have a lower overall communication cost, Cm , while the algorithm solution is $Co > Cm$. This assumption implies that there is a step j in the algorithm such that $Co_{1 \rightarrow j} + Co_{j \rightarrow n} > Cm_{1 \rightarrow j} + Cm_{j \rightarrow n}$, which also implies $Co_{1 \rightarrow n-1} + Co_{n-1 \rightarrow n} > Cm_{1 \rightarrow n-1} + Cm_{n-1 \rightarrow n}$, where the notation $C_{i \rightarrow j}$ means the cost of mapping application stages $i \dots j$. However, this assumption is in contradiction with line 16 of the algorithm that a minimal cost is selected. Thus, the assumption that there is a smaller cost is false and so is the assumption of a lower communication cost. Jones et al. [18] conclude that it is challenging to devise a scheduling function when no a priori knowledge about a job's communication is available. The function used for this work ensures that the minimal number of inter-cluster communication links are introduced when mapping a job to a set of Grid resources. A detailed analysis of cost functions that ensure the optimal execution is beyond the scope of this paper.

5 Experimental Results

The Grid testbed used in our work is the DAS-2 [13] environment, a wide-area distributed system consisting of 200 Dual Pentium-III computer nodes. The environment is built out of clusters of workstations which are interconnected by SurfNet, the Dutch university Internet backbone for wide-area communication, whereas Myrinet, a popular multi-Gigabit LAN, is used for intra-cluster communication. The clusters are located at five Dutch Universities and from this point of view it can be considered as an experimental Grid system operating in the Netherlands.

To quantify the achievement of our proposed enhancements to the KOALA scheduler for HOC-based applications, we measure the improvements in resource utilization and application response time by means of simulations. Second, we show that we are able to successfully schedule HOC-based applications on the DAS-2 by means of KOALA in a transparent way for the end user, as well as the performance improvements achieved by means of our BWCF scheduling algorithm.

The experiments were performed by means of synthetic applications that performed computations and communications according to the three patterns presented in Fig. 1.

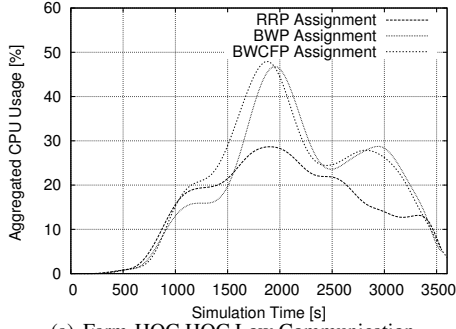
5.1 HOC Scheduling Simulation Results

For the simulation studies, we take each of the currently available HOC implementations (Farm-HOC, Pipeline-HOC and Wavefront-HOC) and schedule it by means of four main policies: round-robin (RR - select sites circularly) assignment, close-to-file (CF) assignment, bandwidth-aware (BW) assignment and bandwidth-aware and close-to-file (BWCF) assignment. The results are presented in Figures 4(b), 4(c) and 4(d).

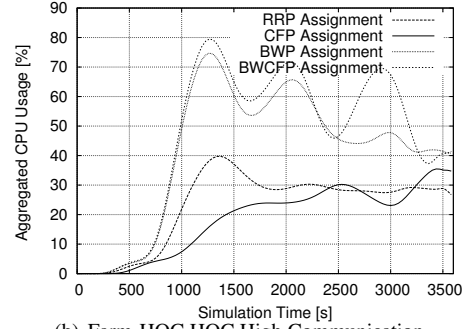
The simulations were performed by means of the GangSim simulator [9], a custom simulator for Grid scheduling studies. As workloads, we considered synthetic HOC applications running on a Grid composed of 20 sites and with the HOC-typical communication behavior specified via input/output file dependencies.

For each HOC, we simulated multiple applications represented by a new data set, being uploaded into the system every 100 seconds. Thereby we simulate a constant arrival rate.

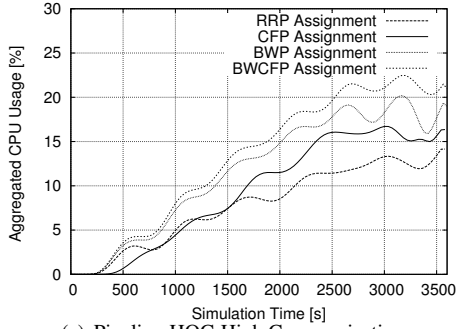
First, we simulated the case of a set of Farm-HOC applications with low communication requirements (compared to the available bandwidth). These results are captured in Fig. 4(a), which shows that the employed scheduling policy practically does not affect the overall application execution. Second, we moved to the case of similar workloads, but with high communication requirements compared to the network links. The results of this scenario show that whenever



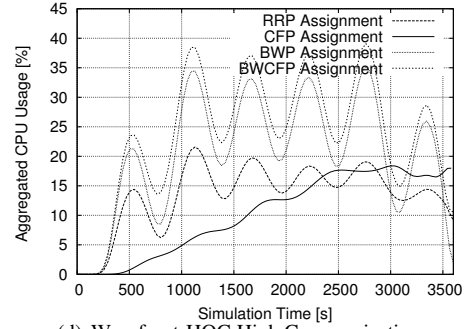
(a) Farm-HOC HOC Low Communication



(b) Farm-HOC HOC High Communication



(c) Pipeline-HOC High Communication



(d) Wavefront-HOC High Communication

the sites with higher bandwidth are chosen, the resource utilization increases by 20% to 50%. At the same time, the performance achieved for the end users (throughput) increases by a factor of 1.5 to 2.

In the figures, a fluctuating execution time can be observed when the number of applications being executed is increased. In the Farm-HOC, this fluctuation is explained by the varying wait times at the workers during the initial data splitting. A similar fluctuation is observable in the Wavefront-HOC case, where some stages must wait for previous stages to finish when there is not sufficient workload pushed into the system to compensate the low utilization intervals.

5.2 HOC Scheduling Experimental Results

The results shown so far were related to simulations running locally on only one machine. We now report on our performance results for employing our proposed scheduling policy on a real Grid testbed. Besides the image processing application [11], which was already mentioned as an example in Section 4.1, we ran synthetic applications also on the Grid, to test the 3 different kinds of available HOCs. As our Grid testbed, we used the DAS-2 system employing KOALA for performing the scheduling.

Tables 1 and 2 capture the performance for the three available types of HOCs. While the strategy employed by the MDRunner (BWCF) can also be applied for scheduling, for the results reported in this paper, it was used to remap, whenever possible, the stage pairs requiring higher communication onto the already reserved nodes with faster communication links. To express this strategy, we used a very simple cost function in our experiments; namely, each intra-cluster link was associated with a cost of 1 unit, while inter-cluster links were associated with a cost of 3 units.

Table 1: Low-Communication Util Metric

Scheduling Policy	HOC Type		
	<i>Far -HOC</i>	<i>Pipeline-HOC</i>	<i>avefront-HOC</i>
CF	64%	91%	97%
BWCF	65%	91%	97%

Table 2: High-Communication Util Metric

Scheduling Policy	HOC Type		
	<i>Far -HOC</i>	<i>Pipeline-HOC</i>	<i>avefront-HOC</i>
CF	22%	83%	79%
BWCF	23%	84%	85%

The experiments were performed for two different scenarios, similarly to the simulation ones in the previous section. The performance metric considered is the ratio of computation time to the total execution time averaged over all HOCs (Util [10]). The higher the value of the Util metric, the better the scheduling strategy. The first set of results reports the Util metric when low communication is required, while the second set reports the same Util metric for 10^4 higher communication requirements. As before, the low communication scenario shows no big difference between the CF and BWCF assignments. However, in the higher communication scenario, the performance improvement due to BWCF becomes noticeable for all HOCs in terms of 1% to 5% lower times spent in communications. We are planning to conduct longer running tests in the future, but currently the DAS-2 does not allow any job to exceed a time limit of 15 minutes.

6 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. This approach is the closest to the work proposed here, the application performance model being described in a high-level language and afterwards mapped by a specialized compiler. The difference in our approach is that we don't use a compiler which may have stalled information about the system status. Moreover, KOALA can also handle changes and re-mappings when the environment changes.

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. We consider monitoring information as an important factor for scheduling, since on the Grid, changes of the platform often happen dynamically.

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.

In the larger context of bandwidth-aware co-scheduling, Jones et al. [18] introduce several bandwidth scheduling policies. They conclude that it is challenging to devise an algorithm when no a priori knowledge about an application communication is provided. Their simulation results show that co-scheduling a large part (85%) of an application on a single cluster provides the best solution in most cases.

7 Conclusions and Future Work

In this paper, we address the problem of transparently scheduling structured parallel applications in Grid environments. Our approach is to enhance Grid scheduling by using the fact that additional knowledge about the application built of pre-implemented components is available at compile time. The particular contributions are as follows:

- We define the four key functionalities for a Grid scheduling infrastructure that targets structured applications. We identify three architectural layers for handling the identified functionalities, namely: translation, resource

management module, and application progress observation.

- We develop and implement the three-layer scheduling architecture for applications based on HOCs (Higher-Order Components). The main feature of our approach is user-transparency: the user is freed from specifying the application behavior and detailed resource requirements. Our user-transparent scheduling covers the whole spectrum of applications for which a performance model can be extrapolated in an automated way.
- We introduce a novel scheduling algorithm (BWCF) that selects the optimal set of Grid sites for execution of a structured parallel application, with respect to a given communication cost function.
- We use a real application example, our pipelined image computation [11], to illustrate how the three abstraction layers are composed for scheduling HOC-based applications.
- The experimental results show the sufficiency of the proposed architectural design and a competitive performance of scheduling on a real Grid testbed.

As future work, we intend to study other types of HOCs and to continue the improvement of scheduling for such applications on Grids. We plan to extend our results to other types of structured parallel applications, with minimal changes of the infrastructure if the performance model cannot be extracted automatically from the application. Also, we will study more realistic workloads and the introduction of predictions about the environment changes.

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, November 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] Anca I. D. Bucur and Dick 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] Jason Carolan, Paul Strong, Ed Turner, and Scott Radeztsky. *Building N1 Grid Solutions: Preparing, Architecting, and Implementing Service-Centric Data Centers*. Sun BluePrints Program. Prentice Hall, 2004.
- [5] B. Cherkassky, A. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. In *ACM-SIAM Symposium on Discrete Algorithms (Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1994.
- [6] M. I. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation*. Research Monographs in Parallel and Distributed Computing. Pitman, London, 1989.
- [7] 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.
- [8] A. Dan, C. Dumitrescu, and M. Ripeanu. Connecting client objectives with resource capabilities: an essential component for grid service management infrastructures. In *The 2nd international conference on Service oriented computing*, 2004.
- [9] C. Dumitrescu and I. Foster. GangSim: A simulator for grid scheduling studies. In *Cluster Computing and Grids*, 2005.
- [10] C. Dumitrescu and I. Foster. GRUBER: A Grid Resource Usage SLA BrokER. In *Proc. of 11th International Euro-Par Conference, Portugal*, 2005.

- [11] Jan Dünnweber, Sergei Gorlatch, Anne Benoit, and Murray Cole. Integrating MPI-Skeletons with Web services. In *Proceedings of the International Conference on Parallel Computing, Malaga, Spain*, September 2005.
- [12] Jan Dünnweber, Sergei Gorlatch, Sonia Campa, Marco Danelutto, and Marco Aldinucci. Using code parameters for component adaptations. In Sergei Gorlatch, editor, *Proceedings of the CoreGRID Integration Workshop, Pisa, Italy*, November 2005.
- [13] Dutch University Backbone. The distributed ASCI supercomputer 2 (DAS-2), 2006.
- [14] Ian Foster, J.-S. Vöckler, M. Wilde, and Y. Zhao. Chimera: A Virtual Data System for Representing, Querying and Automating Data Derivation. In *14th International Conference on Scientific Database Management, Edinburgh*, 2002.
- [15] Sergei Gorlatch and Jan Dünnweber. From Grid Middleware to Grid Applications: Bridging the Gap with HOCs. In *Future Generation Grids*. Springer Verlag, 2005.
- [16] H. Bal et al. Virtual Laboratory for e-Science, 2002.
- [17] Marty Humphrey, Glenn Wasson, Jarek Gawor, Joe Bester, Sam Lang, Ian Foster, Stephen Pickles, Mark Mc Keown, Keith Jackson, Joshua Boverhof, Matt Rodriguez, and Sam 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] W. M. Jones, L. W. Pang, D. Stanzione, and W. B. Ligon III. Bandwidth-aware co-allocation meta-schedulers for mini-grid architectures. In *International Conference on Cluster Computing (Cluster 2004)*, 2004.
- [19] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS)*, pages 104–111, Washington, DC, 1988. IEEE Computer Society.
- [20] A. Mayer, S. McGough, and N. Furmento. ICENI: Optimisation of component applications within a grid environment. In *Parallel Computing Amsterdam*, 2002.
- [21] H.H. Mohamed and D.H.J. Epema. The design and implementation of the KOALA co-allocating grid scheduler. In 640-650 LNCS 3470, editor, *Proceedings of the European Grid Conference, Amsterdam*, 2005.
- [22] K. Ranganathan and A. Dan. Proactive management of service instance pools for meeting service level objectives. In *3rd International Conference on Service Oriented Computing (ICSOC)*, 2005.
- [23] Alain Roy. *End-to-End Quality of Service for High-End Applications*. PhD thesis, The University of Chicago, August 2001.
- [24] The Globus Team. The dynamically-updated request online coallocator DUROC v0.8, 2005.
- [25] The Globus Team. The Globus resource specification language RSL v1.0, 2005.
- [26] N. Tonello, R. Yahyapour, and P. Wieder. A proposal for a generic grid scheduling architecture. In Sergei Gorlatch, editor, *Proceedings of the CoreGRID Integration Workshop, Pisa, Italy*, November 2005.
- [27] Kees van Reeuwijk, Rob V. van Niewpoort, and Henri E. Bal. Developing Java Grid applications with Ibis. In *Proc. of the 11th International Euro-Par Conference*, pages 411–420, Lisbon, Portugal, September 2005.