Reusable cost-based scheduling of grid workflows operating on higher-order components

Reusable Cost-based Scheduling of Grid Workflows Operating on Higher-Order Components

C. L. Dumitrescu and D.H.J. Epema
Technical University of Delft
CoreGRID Institute on Resource Management and Scheduling
e-mail: {C.L.Dumitrescu,D.H.J.Epema}@tudelft.nl

J. Dünnweber and S. Gorlatch
The University of Münster
CoreGRID Institute on Programming Models
e-mail: {duennweb,gorlatch}@uni-muenster.de

Abstract

Grid applications are increasingly being developed as workflows built of well-structured, reusable components. We develop a user-transparent scheduling approach for Higher-Order Components (HOCs) – parallel implementations of typical programming patterns, accessible and customizable via Web services. We introduce a set of cost functions for a reusable scheduling: when the workflow recurs, it is mapped to the same execution nodes, avoiding the need for a repeated scheduling phase. We prove the efficiency of our scheduling by implementing it within the KOALA scheduler and comparing it with KOALA's standard Close-to-File policy. Experiments on scheduling HOC-based applications achieve a 40% speedup in communication and a 100% throughput increase. [1]

1 Introduction

Nowadays, Grid applications are increasingly not developed from scratch: because of their complexity and scale, they rely on pre-packaged pieces of software which are called components, modules, templates, etc. in different contexts. Applications based on the component technology expose a well-defined structure and enable the reuse of code. At the same time, component-based development brings a change of focus for scheduling, which now can make use of the fact that a Grid application communication behavior adheres to well-defined patterns. In traditional scheduling, the user must provide detailed information about the application to be executed, including a precise description of all input files, the executables and the jobs' requirements concerning the runtime platform. The frequency of communication and the amounts of data being communicated strongly matter in Grid applications [14], but they cannot be foreseen by the scheduler before data processing starts and the monitoring system provides feedback.

This paper suggests a scheduling approach, in which the necessary information for the scheduler is provided by the underlying component framework, in a user-transparent manner. We use Higher-Order Components (HOCs [11]) – reusable components that are customizable for particular applications using parameters which may be either data or code. HOCs include the configuration required to run on top of a standard Grid middleware [13] and can be remotely accessed via Web Services. Thereby, HOCs abstract over the technical features of the Grid platform and allow their users to concentrate on their applications.

We use the KOALA scheduler [15] for handling the simultaneous reservation (co-allocation) of the required resources based on the three-layer infrastructure proposed in [5]. Using the BWCF (Bandwidth-Aware Close-to-File) scheduling algorithm introduced in [5], a HOC-based application can be scheduled transparently to the user. The BWCF algorithm is an improvement of the default Close-to-File policy (CF) of KOALA: whereas CF always chooses the minimal transfer paths for exchanging input and output files, BWCF relies on cost functions that take into account bandwidth variations for different execution nodes and the varying communication requirements of different components.

We address the following three questions: (a) What are the most suitable cost functions to assign workflows operating on HOCs to different kinds of resources? (b) If an optimal mapping for a HOC-based workflow is found, then how can it be reused for other applications running the same workflow? and (c) What is the impact of resource usage restrictions, expressed via usage service level agreements (uSLAs [6]) on the performance of HOCs? Besides the user-transparent provision of scheduling information, our main contributions are as follows: first, the identification of the specific factors influencing the scheduling of our HOC-based applications; second, the expression of these factors by means of cost functions that can be integrated with our

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

Proceedings of the Second IEEE International
Conference on e-Science and Grid Computing (e-Science'06)
0-7695-2734-5/06 $20.00 © 2006 IEEE

BWCF scheduling algorithm and the KOALA scheduler; third, the introduction of a schedule reuse technique for similar workflows; fourth, the analysis of controlled resource sharing (resources being provided under pre-defined uSLAs) and its impact on scheduling; and fifth, experiments with the proposed scheduling enhancements on our Grid testbed, the DAS-2 platform [10].

Section II presents HOC-based Grid programming and our proposed cost functions for scheduling. Section III describes the KOALA scheduler and the DAS-2 Grid platform, and shows how we use cost functions for scheduling workflows. Section IV is about restrictions on Grid resources expressed via uSLAs, and their influence on scheduling. Section V presents experimental results, and Section VI concludes the paper by comparing it with related work.

2 Grid, HOCs and Cost Functions

In the following, we describe our Grid environment and identify six cost functions reflecting the communication requirements of HOCs.

2.1 Environment Description

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

- 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.

We use the DAS-2 Grid system for experiments. This Grid testbed is a *space-shared* environment, where a node is exclusively allocated to only one unit of work at a time;

2.2 HOCs and Workflows

Some of the popular patterns of parallelism supported by HOCs are the farm, the pipeline [8] and the wavefront [9] as shown in Fig. 2. The Farm-HOC is used for running applications without dependencies between tasks. All implementations of the Farm have in common the existence of a *Master* unit, where data is partitioned; the parts are then processed in multiple parallel *Worker* units (labeled `W` in the figure). In the Pipeline-HOC, parallelism is achieved by overlapping units for several input instances. The Wavefront-HOC describes computations advancing as a hyperplane in a multidimensional space (called diagonal sweep in the two-dimensional case).

Software built using HOCs exhibits a communication behavior that fits into a certain finite set of communication patterns. The HOCs' structure allows to automatically create performance models that rely on components' behavioral features, e.g., how often two units of work communicate. HOCs come with a support for running them on a particular Grid middleware platform, including configuration files that specify public interfaces (provided via Web services) and the locations of the code for serving the possible requests to a HOC. Scheduling-relevant information, e. g. , the component type, is also included in the HOC configuration. The

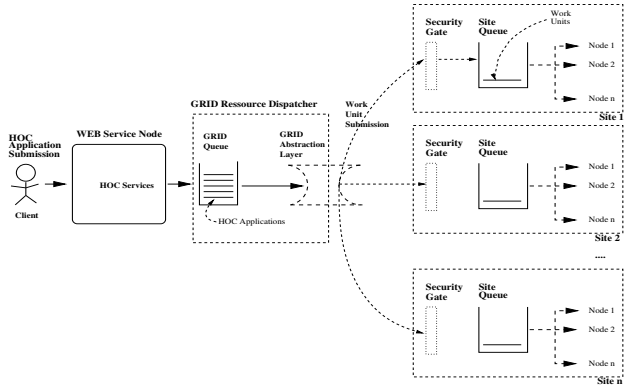

Figure 1. Environment Overview

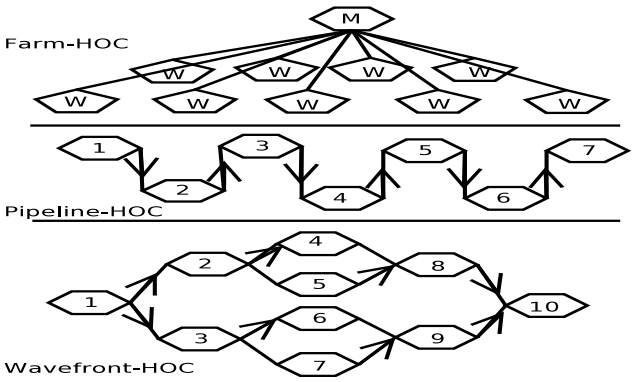

Figure 2. Examples of HOCs

Proceedings of the Second IEEE International
Conference on e-Science and Grid Computing (e-Science'06)
0-7695-2734-5/06 $20.00 © 2006 IEEE

users only upload an application-specific code to a HOC using the respective Web service; they never need to deal with the HOC configuration.

2.3 Influencing Factors

In this section, we identify three generic factors and three HOC-specific factors influencing the performance of the BWCF scheduling algorithm on the DAS-2 system [10]. Our cost functions expect as their parameters various monitoring information which in our case is collected by the KOALA scheduler. The most notable parameters are: (a) the total intra- and inter-site bandwidth (TBW), (b) the available intra- and inter-site bandwidth (ABW), and (c) the application required bandwidth (RBW) measured in megabits per second (Mb/s). The required bandwidth (RBW) for executing an application is represented by the bandwidth requirements of the N work units located on the same execution site ($LC_i, i = 1..N$) and the bandwidth requirements of the M units located on a remote site ($RC_k, k = 1..M$). We compute RBW_j as the sum of the bandwidths of all used links for communications by the j-th unit of work:

$$RBW_j = \Sigma_{i=1}^{N} LC_i + \Sigma_{k=1}^{M} RC_k \quad (1)$$

For each work unit, our scheduling algorithm (BWCF) estimates the time costs of all scheduling possibilities, i. e. , mappings of units onto distributed nodes, and selects the most appropriate ones in terms of a cost function, which is therefore crucial for the efficiency of scheduling. The factors used in our cost functions are the following:

1. **Network Latency:** the co-allocation of resources leads to a data distribution where only a part of the data required by an application is locally available, while other necessary data must be exchanged with remote sites. In our approach, network latency is the sum of the latencies of each node plus the inter-site latencies; both are recorded by the monitoring system and measured in seconds ([s]);
2. **Network Bandwidth:** the interaction of co-allocated parallel applications can create contention in the Grid [14]. To avoid this, our cost functions also incorporate bandwidth data. Based on the monitored bandwidth of each connection, we switch between links whenever a bandwidth improvement is possible. Bandwidth is measured in megabits per second ([Mb/s]);
3. **Network Load:** the load in the network is related to both the frequency of communications and the message sizes. Even if resources exchange small-sized messages, a high amount of such messages may produce a load which requires the choice of nodes with a fast broadband connection. The network load is measured in megabits per second ([Mb/s]).

The next three factors are specific to the pattern-based structure of HOC applications:

1. **HOC Communication Requirements:** the communication requirements of a HOC-based application express how many times each unit of work is required to communicate with other units. If an application performs a lot of local computations and exchanges information only rarely, it has low communication requirements and the three factors listed above will not strongly influence the overall timing. The communication requirements of applications are expressed in megabits per second ([Mb/s]);
2. **HOC Startup Delay:** Running a code unit requires an initialization phase where communication channels are reserved. Since every HOC application requires each work unit to run a particular number of times, we multiply the monitored delay times for data transmissions with a unit factor to build a finer application model. This metric is expressed in seconds ([s]);
3. **HOC Application Layout:** for predicting network load variations accurately, the ordering of the work units in a HOC is also considered as a scheduling factor in our model. Another coefficient is used to weigh unidirectional communication with lower costs. We express this as a set of numbers associated with the application communication behavior measured in megabits per second ([Mb/s]).

A cost function that includes all the above factors is difficult to devise without a precise analysis. Due to the many values expected for each factor in a large Grid, the decision making can become an expensive process.

2.4 The Six Proposed Cost Functions

Cost functions weigh the previously identified requirements in an application-specific way: For some applications, the wall-clock execution times are less important than minimizing the occupancy times of certain resources, while for other applications, the situation is opposite. We introduce the following six cost functions:

1. **Link Count Aware:**

$$F_{link\_count}(j) = N + 3 \times M \quad (2)$$

where N represents the number of neighbor units scheduled on the same site (one network link) and M the number of neighbor units scheduled on other sites (modeled empirically as three network links). When we schedule applications using this function, slower links are weighed with higher costs;

Proceedings of the Second IEEE International
Conference on e-Science and Grid Computing (e-Science'06)
0-7695-2734-5/06 $20.00 © 2006 IEEE

2. **Bandwidth Aware:**

$$F_{bw\_aware}(j) = \Sigma_{i=1}^{N} LC_i/TBW_j + \Sigma_{j=1}^{M} RC_j/TBW_j \quad (3)$$

where LC_i is the bandwidth of the intra-site links used by the unit i, and RC_j is the bandwidth of the inter-site links used by the work unit j, with N and M introduced before. This cost function optimizes over the communication time required for each unit to exchange information with all its peers. Formula (3) sums up the required communication times for local and remote communication, taking network bandwidths into account;

3. **Latency Aware:**

$$F_{lt\_aware}(j) = \Sigma_{i=1}^{N} LC_i/TLT_j + \Sigma_{j=1}^{M} RC_j/TLT_j \quad (4)$$

where TLT stands for network latency ($[s]$). This cost function optimizes over the latency encountered by each unit when exchanging information with all its peers. As in (3), the costs associated with each unit-to-node mapping are weighed by a different factor, which, this time, corresponds to the unit j node's latency;

4. **Network Load Aware:**

$$F_{net\_util}(j) = \Sigma_{i=1}^{N} LC_i^j/ABW_j + \Sigma_{k=1}^{M} RC_k^j/ABW_j \quad (5)$$

Here, the bandwidth factor, which is a constant for the previous cost functions, is replaced by the dynamically monitored bandwidth (ABW_j). In LC_i^j, the index j denotes the work unit number, i. e. , the j-th unit has N local peers and M remote peers, while i stands for the index of the unit's neighbor.

In a HOC-based application, the factors relevant for predicting the time costs of each unit depend on the types of the employed HOCs. Therefore, we propose two additional cost functions, designed for scheduling HOC applications.

1. **Application Topology Aware:**

$$F_{app\_aware}(j) = \Sigma_{l=1}^{j}\Sigma_{i=1}^{N} LC_i^l/ABW_j + \Sigma_{l=1}^{j}\Sigma_{k=1}^{M} RC_k^l/ABW_j \quad (6)$$

This cost function weighs local and remote link capacities as well as the simultaneous communications for all units of an application (given by the employed HOC type and captured by the external summation). Thus, it is probably the most appropriate one for applications with similar requirements for all units, like, e. g. , farm-based ones, while it is less appropriate for pipelined computations that benefit most from the Latency Aware cost function (see experiments);

2. **Predicted Network Variance Aware:**

$$F_{variance}(j) = F_{app\_aware}(j, AL) - PredVar(link\_bw) \quad (7)$$

This cost function incorporates measured network variances and is probably most appropriate for applications with specific requirements for each work unit. It optimizes over both the time required to perform the communication between units and the estimated communication penalties due to the network load produced by other applications. The predictive part in this equation, (7), is measured like described in the work by Yang et al. [17]: any change in the bandwidth usage is considered to repeat in the future, thus, the average of the observed changes are used for computing the prediction.

3 HOC-Aware Scheduling Infrastructure

Due to their purpose of simplifying application development without a significant loss of performance, as compared to hand-tuned applications, HOCs have strict scheduling requirements. We use the KOALA scheduler, whose main features in our context are: (a) user-transparent HOC scheduling [5], (b) multi-HOC application scheduling, and (c) re-use of already computed schedules for HOCs, as detailed in this section.

3.1 The KOALA Co-Allocation Scheduler

KOALA [15] was developed by the PDS group in Delft [16] in the context of the *Virtual Lab for e-Science* (VLe [12]) project. Its main feature is the support for co-allocation, i.e., simultaneous allocation of processors and memory on multiple Grid sites to a single application consisting of several work units.

Our execution testbed is the DAS-2 platform [10], a wide-area network of 200 Dual Pentium-III computer nodes. It comprises sites 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 local communication. The resources in DAS-2 are space-shared: once resources are allocated to a work unit, no other work units can use them.

In our scheduling infrastructure, every submission host has at least one KOALA *runner* in operation, while one single host runs the KOALA *engine*. Runners communicate with the engine via a proprietary light-weight protocol based on exchanging `requests` and `answers`. Applications are executed by submitting single-CPU jobs (e.g., a code parameter for a Pipeline-HOC stage) to runners which handle the reservation of resources.

Proceedings of the Second IEEE International
Conference on e-Science and Grid Computing (e-Science'06)
0-7695-2734-5/06 $20.00 © 2006 IEEE

3.2 The Distribution of Responsibilities

Each HOC execution node runs an MDRunner instance, where `MDRunner` [5] stands for Modified DRunner and DRunner is the KOALA component for scheduling parallel applications via Globus DUROC citeGLOBUS. In this infrastructure, the responsibilities are distributed as follows (see Figure 3):

- 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;
- `MDRunner` makes the scheduling of HOC applications transparent to the user; it generates the HOC application descriptions based on the initial description included in the HOC configuration;
- 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;
- `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;
- BWCF employs different cost functions for optimal resource mapping. The selection of the most appropriate cost function for an application is handled by the `MDRunner` in a user-transparent manner.

3.3 The Multi-HOC Application Support

Compositions of multiple HOCs were not addressed so far, although many applications are based on several different components glued together [2]. In order to schedule applications that use multiple HOCs, we included in the `MDRunner` support for processing workflows, specified via a `MDRunner`-specific workflow description. Such workflows occur when an application invokes several HOCs in a sequential order, for example, the Farm-HOC is employed to distribute the input elements in parallel, after which the application uses the Pipeline-HOC to perform the required data processing.

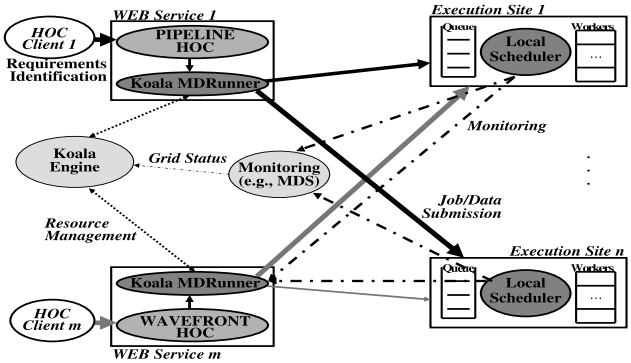

Figure 3. Integration of HOCs with KOALA

Our BWCF scheduling algorithm processes multi-HOC applications by computing the communication costs for each single HOC and by summing up these costs to obtain a scheduling criterion, which is valid for the combination of the employed HOCs without the need for work unit migration.

3.4 HOC Schedule Reuse

Many different applications can be built using the same HOCs or combinations of them, but our automated scheduling of HOC-based applications works independently of application-specific code and data parameters. It is only related to the types of HOCs which are employed and, therefore, different applications using the same HOCs will be mapped onto the same resources. To avoid the need to compute the same schedules again and again, we included in the `MDRunner` the support for reuse of schedules for already processed HOCs and compositions of them.

The scheduling of an application in a Grid environment includes many expensive operations, e. g. , the aggregation of resources for multi-job submissions. In our infrastructure, the reuse of schedules reduces much of the overhead introduced by these operations.

3.5 HOC-Aware Scheduling Algorithm

In our HOC scheduling infrastructure, the `MDRunner` makes use of the BWCF algorithm, which combines the CF scheduling policy with communication cost awareness or other input- and computation-related cost functions. For example, a cost function may take into consideration resource (node or/and network) market values. BWCF replaces the greedy approach of the CF policy by one of the cost functions introduced in Section 2.4 for performing the optimal resource selection.

4 The Impact of uSLAs on HOC Execution

Sharing of resources is required at several levels in multi-domain environments like Grids. Resource sharing becomes challenging, once the distribution of resources spans multiple physical institutions. Because access to resources is controlled, application timeliness becomes important. Resource owners grant the right to use certain amounts of their resources to various consumers. The sharing rules under which resources are made available are expressed using uSLAs [7]. These uSLAs govern the sharing of specific resources among multiple consumers. Once a consumer is

Proceedings of the Second IEEE International
Conference on e-Science and Grid Computing (e-Science'06)
0-7695-2734-5/06 $20.00 © 2006 IEEE

permitted to access a resource via an access policy, the resource uSLA steps in to govern how much of the resource can be consumed [6].

On the DAS-2 system [10], a set of uSLAs are enforced, the most important one being: *any application cannot run for more than 15 minutes from 08:00 to 20:00* and *program execution MUST be performed on the compute nodes, NEVER on a file/compile server*. Such restrictions represent a good motivation for introducing uSLAs for component-based applications, as large scale systems like DAS-2 are a possible target platform for them. Some resource access limitations are not controlled by the scheduler, but via operating system tools (xinetd, tcpd, etc). In its standard implementation, the KOALA scheduler deals with access limitations via trial-and-error: when a Runner reports repeated failures of a job (e. g. , due to insufficient resources) at a site, the site is temporarily removed from the pool of available resources.

We have therefore implemented a *fixed-limit* uSLA mechanism by means of `MDRunner`, ensuring that the resource managers (see Section 3) are not overloaded. In our experiments, the *fixed-limit* uSLA allowed at most four HOC workflows running simultaneously.

5 Experimental Evaluation

In this section, we report the results of experiments with our scheduling approach for three HOC scenarios. In these experiments, we identify empirically the most efficient cost functions for scheduling the different types of HOCs commonly used by means of the BWCF scheduling policy. We use synthetic workloads which well represent the data flows occurring in real applications.

5.1 Performance Metrics

We focus on measuring the performance when employing the reusable workflow scheduling technique. In order to quantify our results, we employ three metrics as follows:

1. *Utilization Ratio* (UR) is defined as the ratio of the computation time and the communication time of a HOC-based application that employs several units that perform a predefined number of computations and require a certain number of message exchanges;
2. *Communication Speedup* (CS) is defined as the ratio of the value of UR for BWCF under a certain cost function to the value of UR for the default KOALA scheduling policy (CF). The formula used to measure the communication improvement for an application that uses a single HOC is:

$$CS = (UR_{CF} - UR_{Cost_Func}) / UR_{CF} \quad (8)$$

where $Cost\ Func$ stands for the analyzed cost function and CF stands for the time costs using KOALA's Close-To-File scheduling policy.
3. *Throughput* is defined as the number of work units executed on the Grid in a pre-defined interval of time.

5.2 Experimental Results

We have performed three sets of experiments for three different scenarios in scheduling HOC-based applications, namely *communication-size analysis*, *cost function analysis*, and *schedule reuse analysis*. The results are captured in Tables 1, 2, and 3. The values in parentheses are the number of units and messages. The values in parentheses in the next two tables are the number of units, the number of exchanged messages and the size of each message.

5.2.1 The Communication-size Analysis Scenario

For the communication-size analysis, we perform 10 runs of each single HOC type using synthetic applications and we present the average value and the standard deviation. Each HOC-based application was composed of 15 to 22 units and ran on three DAS-2 sites: Delft, Utrecht, and Leiden.

Table 1. Speedup for Link-Count (%)

Comm. Req.	Synthetic HOC Application Type		
	Farm (15x20)	*Pipe (15x20)*	*Wave (22x20)*
50Kb	0.71±3.2	10.95±6.6	15.81±15.2
100Kb	0.47±2.2	12.15±7.9	14.11±6.4
500Kb	0.87±4.5	14.68±2.5	14.22±4.6
1Mb	0.19±2.8	13.59±4.9	14.26±2.6
5Mb	0.89±1.4	12.56±5.5	15.38±2.8
10Mb	0.23±5.9	13.61±5.6	14.86±5.6

For the Farm-HOC, we observed similar performance when using *link-count* and the CF policy (see Table 1), which we explain by the identical mapping of units to resources. However, when more resources were available, KOALA's default scheduling policy performs worse as shown in Table 2. For the Pipeline-HOC and the Wavefront-HOC, the performance increase when using the *link-count* is similar, regardless of the amount of input data (the gain is expressed in percentage for 20 data items of 50Kb–10Mb).

5.2.2 The Cost Function Analysis Scenario

For the cost function analysis, each HOC-based application exchanged 20 messages with 2 to 10 Mb of data (a 10 times higher communication requirement than in the previous scenario), while running on all five sites of DAS-2.

Table 2 captures our results for running communication-intensive HOC-based applications: they consist of 20 to 50

Proceedings of the Second IEEE International
Conference on e-Science and Grid Computing (e-Science'06)
0-7695-2734-5/06 $20.00 © 2006 IEEE

units of work and have high communication requirements compared to the capacity of the inter-site network links. We observe that the *link-count* cost function (our previous choice) yields a lower performance than the *bandwidth-aware* cost function, while our implementation of the *application-aware* cost function introduces the lowest performance, even lower than the performance of the default CF scheduling policy (Table 2). The value for the standard deviation is quite high (Table 2), since we compared BWCF also with KOALA's default scheduling policy, which leads to much higher time needs in component-based applications, regardless of the employed cost function.

Table 2. Speedup for the Six Costs (%)

Cost Function	Synthetic HOC Application Type		
	Farm (20×20×2M)	*Pipeline* (50×20×10M)	*Wavefront* (46×20×10M)
Link Count	7.18±9.6	21.46±19.4	34.00±26.9
Bandwidth	34.06±6.8	20.90±19.5	39.37±29.2
Latency	31.06±2.8	24.60±19.2	28.30±29.2
Network	33.80±6.8	22.60±19.1	17.40±33.6
Application	36.10±5.9	-5.00±0.1	-1.13±4.6
Predicted	34.21±1.8	21.04±19.5	38.22±6.2

In summary, the most appropriate cost function seems to be the *bandwidth-aware* function. By HOC type, the most appropriate functions are *application-aware* for the Farm-HOC, and *latency-aware* for the Pipeline-HOC, which supports our assumption. For the Wavefront-HOC, the *bandwidth-aware* and *predicted* cost functions lead to the most efficient scheduling.

5.2.3 The Schedule Reuse Analysis Scenario

In the schedule reuse analysis, we test the performance of our scheduling when schedules are reused for HOC-based applications that exhibit the same workflow, i.e., they employ the same HOCs in identical order. Also, a *fixed-limit* uSLA at the user level was enforced. The uSLA allowed at most four applications to run in parallel on the DAS-2 resources. Table 3 captures our results for HOCs with 15 to 22 units and message sizes of 1Mb to 10 Mb.

We note 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.

This observation can be easily traced in Fig. 4, where the work unit termination time is plotted on the vertical axis. Once a HOC instance terminates and is requested again, the reserved resources are reused instead of computing a new schedule.

Table 3. Throughput Gains with Reuse (%)

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

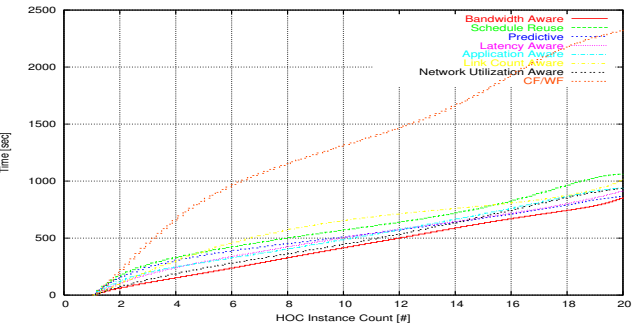

Figure 4. Farm-HOC Execution Shapes

6 Related Work

The workflow descriptions processed by our `MDRunner` serve a similar purpose as the ASSIST coordination language, devised by Aldinucci et al. [1]. A specialized compiler translates the graph of modules into a network of processes on a Grid, under pre-specified rules. In our approach, applications are never re-scheduled, which is an advantage, compared to ASSIST. Moreover, BWCF takes application submission costs into account and optimizes the scheduling of workflows, which is not the case in ASSIST. Benoit et al. used a Performance Evaluation Process Algebra (PEPA) [4] which can be used for mapping ASSIST applications onto Grid resources [3]. Contrary to this approach, the input to our cost functions is directly given by the monitoring information and no scheduling requirements are described using a specific formalism.

In the broader context of bandwidth-aware scheduling, Jones et al. [14] introduce several scheduling policies for parallel applications where information about the communication behavior is provided by the user. They conclude that it is challenging to devise a scheduling algorithm when no a priori knowledge about an application is provided. Their simulation results show that co-scheduling a large part (85%) of an application on a single site provides the best solution, independently of the communication pattern.

Proceedings of the Second IEEE International
Conference on e-Science and Grid Computing (e-Science'06)
0-7695-2734-5/06 $20.00 © 2006 IEEE

7 Conclusions

In this paper, we addressed the problem of user-transparent scheduling for application workflows built out of higher-order components (HOCs). We introduced our approach for enabling the KOALA Grid scheduler to perform communication cost-aware scheduling by using the knowledge about an application behavior available at compile time. We experimentally proved advantages over KOALA's CF scheduling for three advanced scheduling techniques, namely: cost-based scheduling, multi-HOC workflow support, and schedule reuse.

We focused on identifying the specific factors that must be considered for HOC-based application scheduling. In a set of experiments, we addressed the question "*What are the most suitable cost functions for scheduling workloads of HOC-based applications?* " The answer is: the *bandwidth-aware* and the *predicted-variance* functions. The improvements for single large HOC-based applications and for multi-HOC workflows surpassed the other cost functions by 5% to 20%. The aggregated submission provided additional gains by reducing the submission time for multi-component applications, and the schedule reuse led to the highest performance improvement, whenever workflow repetitions allowed to apply it.

Another problem addressed in this paper is how resource policies (uSLAs) affect HOC scheduling. Our experiments proved that uSLAs, in combination with workflow aggregation and schedule reuse, do not impede on the overall performance gains. Future work will study the impact of the monitoring quality on the proposed scheduling infrastructure, exploration of alternative scheduling heuristics and extensions of KOALA for preventing applications from conflicting with each other.

Acknowledgments

We would like to thank the DAS-2 system team for kindly providing their resources for our experiments.

Sincere thanks also go to the anonymous referees of the draft version of this paper for their constructive comments.

References

[1] M. Aldinucci, M. Danelutto, and M. Vanneschi. Autonomic QoS in ASSIST Grid-aware Components. In *Euromicro PDP 2006: Parallel Distributed and network-based Processing, IEEE*, Montbliard, France, 2006.

[2] M. Alt, S. Gorlatch, A. Hoheisel, and H.-W. Pohl. A Grid Workflow Language Using High-Level Petri Nets. In *Second Grid Resource Management Workshop*, Poznan, Poland, September 2005.

[3] 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.

[4] 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 (PAPP'04)*, 2005.

[5] 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*, Paris, France, 2006.

[6] C. Dumitrescu, I. Raicu, and I. Foster. Experiences in running workloads over Grid3. In *Proceedings of Grid and Cooperative Computing (GCC'05)*, Beijing, China, 2005.

[7] 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, Stockholm, Sweden, June 2005.

[8] 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.

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

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

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

[12] H. Bal et al. Virtual Laboratory for e-Science (vl-e). Web Page: http://www.vl-e.nl, 2002.

[13] 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 *Proceedings of the 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, Triangle Park, North Carolina, 2005.

[14] W. M. Jones, L. W. Pang, D. Stanzione, and W. Ligon III. Bandwidth-aware co-allocation meta-schedulers for mini-grid architectures. In *International Conference on Cluster Computing (Cluster 2004)*, San Diego, California, 2004.

[15] H. Mohamed and D. Epema. The Design and Implementation of the KOALA Co-Allocating Grid Scheduler. In *Proceedings of the European Grid Conference, Amsterdam*, volume 3470 of *LNCS*, pages 640–650, 2005.

[16] KOALA Co-Allocating Grid Scheduler. Web Page: http://www.st.ewi.tudelft.nl/koala/.

[17] L. Yang, J. Schopf, and I. Foster. Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments. In *Proceedings of the International Conference for High Performance Computing and Communications (SC'03)*, Phoenix, Arizona, 2003.

Proceedings of the Second IEEE International
Conference on e-Science and Grid Computing (e-Science'06)
0-7695-2734-5/06 $20.00 © 2006 IEEE