Scheduling of Scientific Workflows in the ASKALON Grid Environment*

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

ABSTRACT

Scheduling is a key concern for the execution of performance-driven Grid applications. In this paper we comparatively examine different existing approaches for scheduling of scientific workflow applications in a Grid environment. We evaluate three algorithms namely genetic, HEFT, and simple "myopic" and compare incremental workflow partitioning against the full-graph scheduling strategy. We demonstrate experiments using real-world scientific applications covering both balanced (symmetric) and unbalanced (asymmetric) workflows. Our results demonstrate that full-graph scheduling with the HEFT algorithm performs best compared to the other strategies examined in this paper.

1. INTRODUCTION

Scheduling of scientific workflow applications on the Grid is a challenging problem, which is an ongoing research effort followed by many groups. Deelman [9] distinguishes several workflow processing strategies covering trade-offs between dynamicity and look-ahead range in workflow processing. In [3] Deelman proposed a scheduling strategy based on initial partitioning of the workflow into sequential sub-workflows, that are scheduled sequentially one after another. Prodan [10] applied genetic algorithms [7] to schedule the whole workflow at once, and rescheduling it many times during the execution. These approaches were not compared against each other.

In this paper we examine three scheduling algorithms to evaluate their performance for scheduling scientific workflows in Grid environments. The scheduling algorithms comprise a genetic algorithm similar to the one presented in [10], the well-known HEFT algorithm [15], and a "myopic" algorithm. The HEFT algorithm is an extension for heterogeneous environments of the classical list scheduling algorithm [8]. HEFT is a simple and computationally inexpensive algorithm, which schedules workflows by creating an ordered list of tasks out of the workflow, and mapping the tasks to the resources in the most appropriate way. Execution order is based on the list created in the first two phases of the algorithm. The last algorithm we applied is a simple "myopic" algorithm, similar to the Condor DAGMan [12] resource broker, which schedules the next task onto the best machine available without any long-term optimization strategy. The Grid model applied by us in the experiments assumes high availability rate and good control over the resources by the scheduler. This is not always assumed by many other Grid research groups, but it is usually the case for scientific workflows executed in research institutions.

Additionally, we compared different scheduling strategies including full graph scheduling and incremental workflow partitioning strategy [3]. The Myopic algorithm can be considered as a just-in-time scheduling strategy, as the scheduling decisions made by the algorithm are optimized for the current time instance.

In the remainder of this paper, we evaluate the scheduling approaches through a series of experiments. We show, that the HEFT algorithm is more effective and less time consuming than the genetic algorithms applied in [10]. HEFT also performs substantially better than a simple Myopic algorithm. We also show that the workflow partitioning approach described in [3] does not appear to imply any advantage over full-graph scheduling. For the class of strongly unbalanced (asymmetric) workflows we highlight poor performance of the incremental workflow partitioning and simple scheduling algorithms. Full-ahead scheduling with the HEFT algorithm appears to perform best for unbalanced workflows.

*This research is partially supported by the Austrian Science Fund as part of the Aurora project under contract SFBF1104 and the Austrian Federal Ministry for Education, Science and Culture as part of the Austrian Grid project under contract GZ 4003/2-VI/4c/2004.

2. ASKALON ENVIRONMENT

ASKALON [5] is a Grid environment for composition and execution of scientific workflow applications. The workflow model adopted in ASKALON is described in Section 3. The scheduler optimizes for performance using the execution time as the most important goal function. The scheduler interacts with the enactment engine (see Fig. 1) which is a service that supervises the reliable and fault tolerant execution of the tasks and transfer of the files. The resource broker and the performance predictor are auxiliary services which provide information about the resources available on the Grid, and predictions about expected execution times and data transfer times. The performance monitoring service provides up-to-date status information of the application execution and of the Grid environment. This information can be used by the scheduler to make a decision about rescheduling.

The scheduler itself consists of several components. The workflow evaluator transforms the dynamic and compact representation of the workflows in a static structure as described in Sec. 3. The scheduling engine performs the actual scheduling, applying one of the alternative scheduling algorithms. The event generator is meant for generation of rescheduling events to cope with the dynamic nature of workflows and the Grid and is currently being implemented.

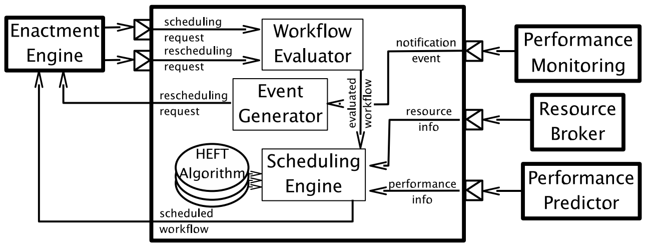


Figure 1: ASKALON environment architecture

3. WORKFLOW MODEL

Scientific workflows executed in the ASKALON environment are based on the model described in the AGWL specification language [13]. AGWL documents can express simple DAGs as well as more sophisticated workflow graphs containing loops and conditional branches which impose control flow decisions that can only be decided at runtime. The condition of a conditional branch (either *if-then* or *switch*) may be evaluated in various ways for different executions, or a *while-loop* may have different number of iterations. Furthermore, *parallel-for* loops are introduced to specify a large number of parallel activities (hundreds) in a compact form, for scalability reasons. Such parallel-for constructs may be evaluated differently at run-time, depending on the parameters of the current execution. The actual number of parallel activities specified by a parallel-for construct may not be known at the beginning of the workflow execution. In order to apply a full-graph scheduling algorithm, all such uncertainties have to be resolved. To this end, we make assumptions about the actual evaluation of the control structures. If an assumption fails, the scheduler transforms the workflow once again in the proper way and reschedules it. This approach may bring considerable benefit if the structure of the workflow is predicted correctly (especially, when a strong unbalance in the workflow is detected). If the conditions are predicted incorrectly, the workflow execution time is the same as in the case of a just-in-time strategy which schedules only those parts of the workflow that are resolved at the moment of scheduling. Fig. 5-8 show two such workflow transformations applied to real Grid workflow applications (see Section 6).

4. SCHEDULING ALGORITHMS

The scheduling algorithms under consideration map tasks as part of workflows onto Grid sites (clusters). Each Grid site consists of a set of CPUs, each of which is considered as a single computational resource. If a task is executed on a CPU, no other tasks can use the same CPU at the same time. Execution times of tasks and data transfers generated by the performance predictor are given as input data to the scheduler.

4.1 HEFT algorithm

The HEFT algorithm that we applied consists of 3 phases:

1. *Weighting* assigns the weights to the nodes and edges in the workflow;
2. *Ranking* creates a sorted list of tasks, organized in the order how they should be executed;
3. *Mapping* assigns the tasks to the resources.

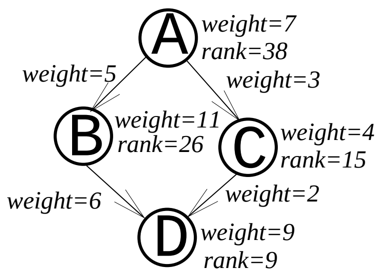
The weights assigned to the nodes are calculated based on the predicted execution times of the tasks. The weights assigned to the edges are calculated based on predicted times of the data transferred between the resources. In homogeneous environments the weights are equal to the predicted times. In heterogeneous environments, the weights must be approximated considering different predictions for execution times on different resources, and for different data transfer times on different data links. Several approximation methods were proposed and compared [11]. Each of them provides different accuracy for different cases. We chose the arithmetic average.

The ranking phase is performed traversing the workflow graph upwards, and assigning a rank value to each of the tasks. Rank value is equal to the weight of the node plus the execution time of the successors. The successor execution time is estimated, for every edge being immediate successors of the node, adding its weight to the rank value of the successive node, and choosing the maximum of the summations. A list of resources is arranged, according to the decreasing rank values. An example workflow graph with the calculated weights and ranks is shown in Fig. 2. The example considers 3 heterogeneous resources R1, R2 and R3. Data transfer is assumed to be equal in both directions between any two of those resources.

In the mapping phase, consecutive tasks from the ranking list are mapped to the resources. For each task, the resource which provides the earliest expected time to finish execution is chosen. The pseudocode of the HEFT algorithm is depicted in Alg. 1.

4.2 Genetic Algorithms

Genetic Algorithms are a part of evolutionary computing, inspired by Darwin's theory of evolution. They represent powerful optimization heuristics, used to search global minima in multi-dimensional search spaces. The basis of the algorithm is to encode possible solutions of the problem into a *population* of *chromosomes*, and subsequently to transform the population using standard operations of *selection*, *crossover* and *mutation*, producing successive *generations*. The selection is driven by an established *fitness function*, which evaluates the chromosomes in terms of accuracy of the represented solutions. Crossover and mutation respond



	R1	R2	R3	*avg*
A	5	8	8	7
B	9	13	11	11
C	3	4	5	4
D	7	10	10	9

execution times on different resources

	R1−>R2	R1−>R3	R2−>R3	*avg*
A−>B	6	4	5	5
A−>C	4	2	3	3
B−>D	7	4	7	6
C−>D	1	1	4	2

data transfer time between different resources

Figure 2: Weights and ranks calculated with HEFT algorithm

algorithm 1 HEFT algorithm

```
$T$ - set of all tasks in the workflow,
$E$ - set of all dependencies in the workflow,
$R$ - set of all available resources,
$(t_1, t_2)$ - dependence between tasks $t_1$ and $t_2$
$time(t, r)$ - execution time of task $t$ on resource $r$,
$time(e, r1, r2)$ - data transfer time of data between
resources $r1$ and $r2$ (dependence $e$ in the workflow),
## Weighting phase##
for each $t \in T$ do
~  $w(t) = \frac{\sum_{r \in R} time(t,r)}{\#R}$
for each $e \in E$ do
~  $w(e) = \frac{\sum_{r_1, r_2 \in R, r_1 \neq r_2} time(e, r_1, r_2)}{\#R \cdot (\#R-1)}$
##Ranking phase##
$Succ = \{(t_1, t_2) : t_1, t_2 \in T \wedge (t_1, t_2) \in E\}$
$Nsucc = Succ$
$NT = T$
while $NT \neq \{\}$ do
~  $Last = \{t : t \in NT \wedge \neg \exists t_1 : (t, t_1) \in NSucc\}$
~  for each $t \in Last$ do
~     $LS(t) = \{t_1 : (t, t_1) \in Succ\}$
~     $rank(t) = w(t) + max(\{0\} \cup \{r : r = w(t_1) + w((t, t_1)) \wedge t_1 \in LS(t)\})$
~     $NSucc = NSucc \setminus \{(t_1, t) : (t_1, t) \in NSucc\}$
~  end
~  $NT = NT \setminus Last$
end
$ranking\_list = sort(T, rank)$
##Mapping phase##
for $i = \#ranking\_list$ downto 1 do
$t = ranking\_list[i]$
~ Find resource $r \in R$ : $finish\_time(t, r)$ is $min$;
~ Schedule $t$ to $r$;
~ Mark $r$ as reserved until $finish\_time(t, r)$;
end
```

to standard biological operations of mutual exchange of a part of body within a pair of chromosomes, and of change of some elements (so-called *genes*) in the chromosomes randomly selected from the population. The end condition of a genetic algorithm is usually the *convergence criterion* which checks how much the best individual found changes between subsequent generations. A maximum number of generations can also be established. The pseudocode of a genetic algorithm is presented in Alg. 2.

algorithm 2 Genetic algorithm

```
 Create the initial population of chromosomes;
while convergence criteria is false do
~ Perform crossover and mutation;
~ Calculate fitness values for the population;
~ Create a new population, based on actual fitness
values;
end
```

Genetic Algorithms are a good general purpose heuristic, which is able to find the optimal solution even for complicated multi-dimensional problems. By transforming a broad population of chromosomes in a semi-random manner, the entire search space is traversed and the search does not end up in a local minimum. However, Genetic Algorithms are not equally appropriate for every possible optimization problem. Solutions of the problem must be properly encoded into the chromosomes, which is not always feasible. Prodan in [10] encoded the actual mapping of tasks to the resources without specifying the order of execution of independent tasks (not linked through control and data flow dependencies) that are scheduled on the same CPU. Therefore this execution order cannot be a subject to optimization. Moreover, Genetic Algorithms tend to be computationally extensive.

4.3 Myopic algorithm

To compare the scheduling algorithms described so far, we developed a simple and inexpensive scheduling algorithm, which makes the planning based on locally optimal decisions. The algorithm represents a class of schedulers covering for instance the Condor DAGMan resource broker which employs the matchmaking mechanism [12]. The pseudocode of the algorithm is described in Alg. 3.

algorithm 3 Myopic algorithm

```
 $T$ - set of all tasks in the workflow,
$NT = T$
while $NT \neq \{\}$ do
~ Find task $t \in NT$ : $earliest\_starting\_time(t)$ is
$min$;
~ Find resource $r \in R$ : $finish\_time(t, r)$ is $min$;
~ Schedule $t$ to $r$;
~ Mark $r$ as reserved until $finish\_time(t, r)$;
~ $NT = NT \setminus \{t\}$
end
```

The Myopic algorithm can produce reasonably accurate results for rather simple workflows given accurate performance predictions. But it does not provide any full-graph analysis and does not consider the order of task execution.

5. SCHEDULING STRATEGIES

Different scheduling strategies [9] can be applied, considering the trade-off between dynamicity and look-ahead range in workflow processing. *Just-in-time* strategy, in [9] referred to as *in-time local scheduling*, consists of mapping the tasks to the resources, always choosing the most appropriate solution for the current step. This approach benefits from using the most up-to-date performance data, which is important for the Grid, but on the other hand it neglects the graph structure, and the whole workflow may not be scheduled optimally. At the other extreme, we have *full-ahead planning* where the full-graph scheduling is performed at the beginning of execution. In this case, a sophisticated graph scheduling algorithm can be applied, but the dynamism of the Grid is not considered. Intermediate solutions try to reconcile workflow planning with Grid dynamism, and to find an approach which considers both the workflow structure and the Grid behavior. One of the possible solutions is the workflow partitioning applied in the Pegasus system [3]. It consists of an initial partitioning of the workflow into a sequence of subworkflows, which are subsequently scheduled and executed (see Fig. 3).

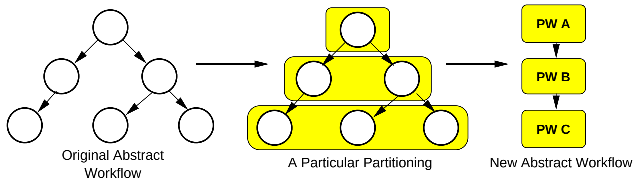


Figure 3: Workflow partitioning in Pegasus [3]

Each partitioning can be characterized by the width of a slice. The width of a slice is expressed as maximal number of node layers within each slice. For instance, the workflow depicted on Fig. 3 was partitioned with one layer per slice (*1-layer partitioning*). Any element of the sequence can be scheduled and executed only if the immediate predecessor has already finished its execution. This approach has an advantage over the simple just-in-time scheduling, as the planner considers more than one task at the time, and has a better overview of the whole graph.

Full-graph scheduling, however, can also be applied in a dynamic way. If we do not consider the initial scheduling as the ultimate decision but only a hint, and if we admit subsequent reschedulings if they are necessary, we can apply a full-graph scheduling algorithm many times during the execution of a workflow. One of the workflows we applied in our experiments belongs to a specific class of strongly unbalanced workflows, which seems to require full-graph analysis for proper scheduling. The workflow contains a parallel section and some of the branches take longer to execute than the others (see Fig. 8). The tasks that belong to the longer branch should, therefore, execute with higher priority than the ones that belong to the shorter branches. One important goal of our experiments was to investigate how the scheduling results depend on the workflow strategy applied for strongly unbalanced workflows.

As our experiments concern scientific workflows executed in research institutions, we assume high availability rate and good control over the resources, what is not always the case for best-effort Grid schedulers. In particular, we assume that the scheduler can have precise information about the resources available in the Grid, and the submissions made by the scheduler are executed as they were requested. We also assume that no failures occur during the execution, so that the execution of the workflow is performed in the same way as it was planned by the scheduler.

6. EXPERIMENTAL RESULTS

In our experiments we compare the HEFT algorithm with a genetic algorithm similar to the one proposed in [10], and with the Myopic algorithm described in Section 4.3. We also compare the full-graph scheduling with the workflow partitioning strategy. As results, we show execution times of the scheduled workflow applications (*execution times*), and the times spent in preparing the schedules (*scheduling times*). The execution times were measured for two scenarios of workflow execution. In the first scenario, we do not provide to the scheduler any performance predictions, so the scheduler has to assume that all the execution times are equal for all tasks on all the resources (scheduling *without performance guidance*). In the second scenario, the scheduler is provided with experience-based performance predictions derived from historical executions (scheduling *with performance guidance*). The predictions were provided to the scheduler in a two-dimensional array, containing the execution time of each task on each computer architecture available in our Grid. The assumption was that each task takes the same execution time on every machine that belongs to the same type (i.e, has the same CPU model, CPU speed and total RAM size).

Experiments were performed incorporating seven Grid sites (clusters) of the Austrian Grid [2] infrustructure with 116 CPUs in total (not all Grid sites were used in all the experiments). In Fig. 4 we present the performance of the individual clusters, where each cluster shows the average execution time of all the workflow tasks executed on a single CPU. As we can see, the fastest cluster is more than three times faster than the slowest one.

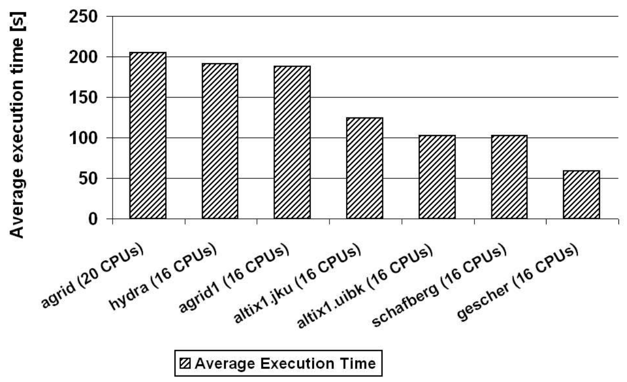


Figure 4: Performance of Grid clusters used in the experiments.

Similarly to execution time predictions, the execution times of the tasks on the sites were measured on the Austrian Grid during a test phase. Time consumed by data transfers between two tasks connected with a data link was considered as constant. We also fixed the middleware overhead introduced by the Globus GSI security [6] and the PBS queuing system [14] to 30 seconds.

We used two real-world workflow applications in our experiments. WIEN2k [1] is a quantum chemistry application developed at Vienna University of Technology. WIEN2k workflow (Fig. 5) is a fully-balanced workflow which contains two parallel sections with possibly many parallel tasks, and an external loop. For our tests we considered the workflow with one iteration of the loop, and 250 parallel tasks in each section (Fig. 6). Invmod [4] is a hydrological application developed at the University of Innsbruck, designed for calibration of parameters for the WaSiM tool developed at the Swiss Federal Institute of Technology Zurich. The Invmod workflow (Fig. 7) consists of an outermost parallel loop with many iterations (executed as separate threads), which contains a nested optimization loop. We used this workflow to simulate the common case of strongly unbalanced workflow. If the loops in individual workflow threads have different numbers of iterations (and the iteration numbers are predicted correctly), then the threads may differ significantly with regard to their expected execution times. The workflow used for these experiments (Fig. 8) contains 100 parallel iterations one of which contains 20 iterations of the optimization loop. The remaining 99 iterations contain 10 optimization iterations each. It means, that one of the threads takes approximately twice as much execution time as all others.

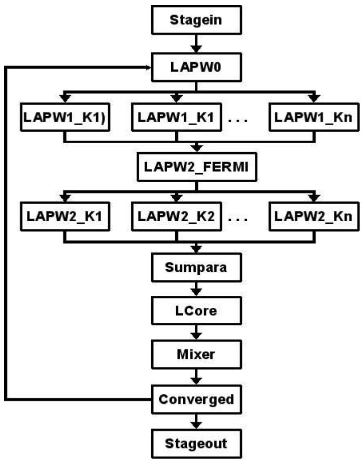


Figure 5: WIEN2k, original workflow.

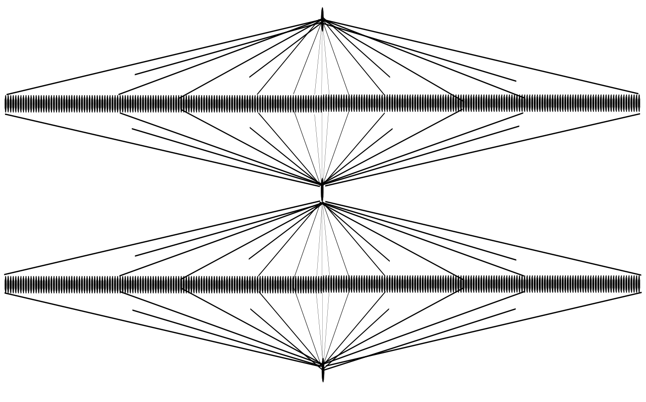


Figure 6: WIEN2k, transformed workflow.

The genetic algorithm that we applied is based on the population of 100 chromosomes transformed in 20 generations, which is not a large number but it allowed us to achieve a good convergence rate with relatively small scheduling time. Probability of crossover was fixed by us to 0.25, and mutation rate to 0.01. We performed workflow partitioning by dividing the workflow into slices with well-defined width (see Section 5). For the WIEN2k workflow (consisting of five layers) we applied a three-layer partioning, and for Invmod workflow (which consists of 44 layers) we applied three different partitionings, with 10, 20 and 30 layers.

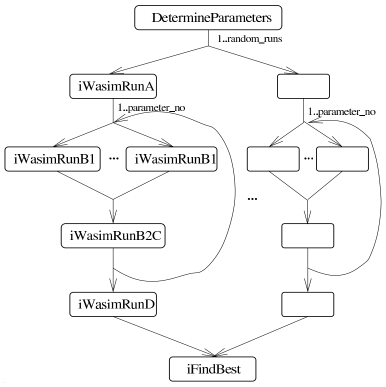


Figure 7: Invmod, original workflow.

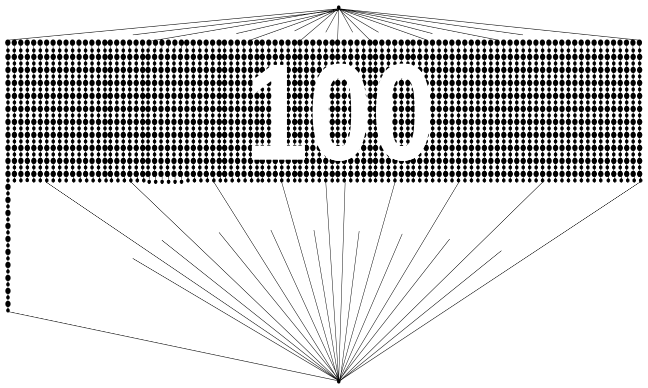


Figure 8: Invmod, transformed workflow.

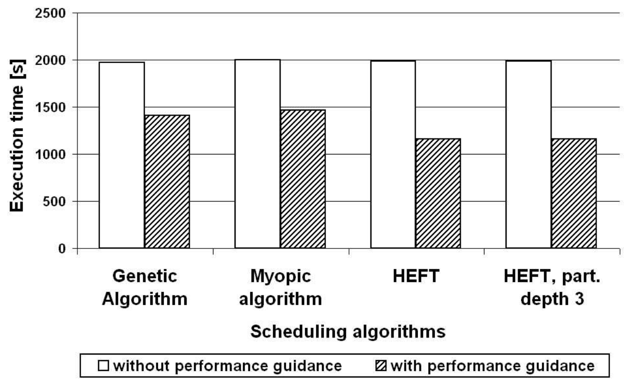


Figure 9: WIEN2k executed in heterogeneous environment, execution time.

The first conclusion we draw from the results (Fig. 9-12) is that performance prediction is very important in heterogeneous Grid environments. For both workflows, the results achieved with performance guidance are in the best case nearly two times better that the results achieved without performance guidance. Performance estimates are clearly important even if they are not highly accurate.

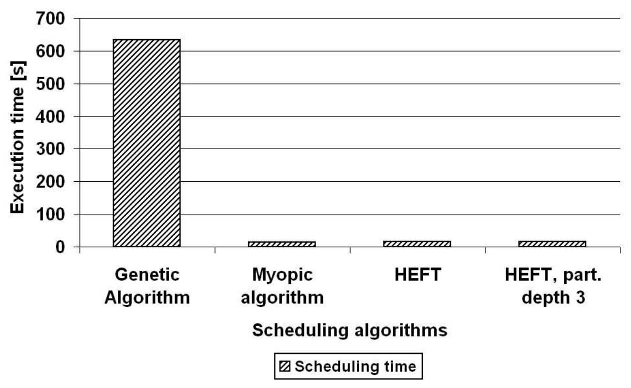


Figure 10: WIEN2k executed in heterogeneous environment, scheduling time.

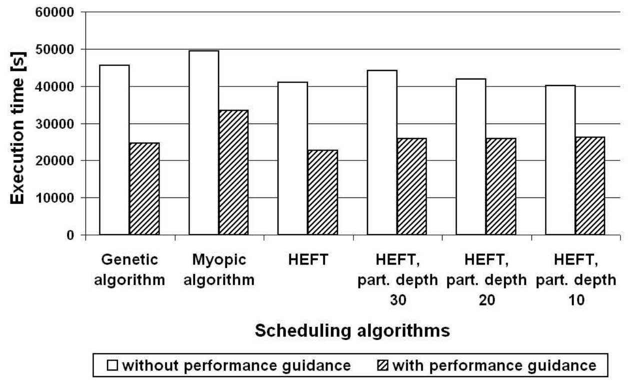


Figure 11: Invmod executed in heterogeneous environment, execution time.

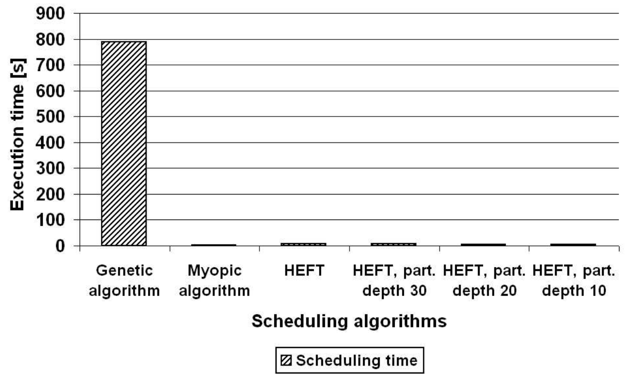


Figure 12: Invmod executed in heterogeneous environment, scheduling time.

Comparing the results measured for the WIEN2k workflow we can notice that HEFT produces much better results than the other algorithms. Execution time of the workflow is 17% shorter than for the genetic algorithm, and even 21% than for the Myopic. The simple solution applied in Myopic appears to be insufficient for large and complicated workflows, and the algorithm produces the worst results. Also the genetic algorithm appears to be not a good method to deal with our problem. It was able to approximate the global maximum, but it did not find the actual best value which lies probably in a "long and narrow corner" of the search space. For the scheduling without performance guidance, where the search space has more regular borders, the genetic algorithm behaves equally good (or even better) than all the other algorithms. Comparing the scheduling times of individual algorithms we can see that the genetic algorithm executes two to three orders of magnitude longer than the others. It means, that even to generate a single population takes much longer than the HEFT algorithm.

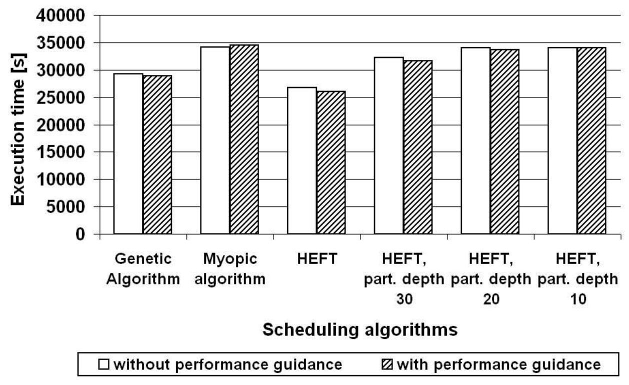


Figure 13: Invmod executed in homogeneous environment, execution time.

The results measured for the Invmod workflow present how individual algorithms deal with strongly unbalanced workflows. As expected, the Myopic algorithm provides the worst results of all, approximately 32% worse than HEFT. The genetic algorithm produces quite good results. It was able to locate the area where the global minimum is located, but it was not able to find the best possible solution, since the order of execution (of independent tasks scheduled to the same CPU) was not considered for optimization. In the workflows scheduled without an established task order, the tasks are executed in an arbitrary order chosen by the runtime system. For a strongly unbalanced workflow, however, the tasks that execute in iterations of the parallel loop with longer execution time should be executed more often than the others, which cannot be done by the runtime system which does not consider global graph structure. Scheduling strategies based on the workflow partitioning were also not able to find the optimal solution, although their results are still better than the one found by the Myopic algorithm. Only the full-graph analysis could find a well performing scheduling solution for imbalanced workflows. Since all of the algorithms (except for the genetic algorithm) execute really fast (less than 20 seconds for large and complicated workflows), there is no reason to apply the partitioning strategy in place of the full-graph analysis.

Fig. 13 presents the execution results of the Invmod workflow on a homogeneous environment (three nearly identical Grid sites). As expected, there is now almost no difference between the scheduling with and without performance guidance, as the execution on each cluster takes the same time. Again, HEFT produces the best results, 24% better than Myopic.

7. CONCLUSIONS AND FUTURE WORK

Scheduling applications on the Grid is of paramount importance to optimize non-functional parameters such as execution time. In this paper we compared three different algorithms examining aspects such as incremental versus full-graph scheduling for balanced versus unbalanced workflows.

Based on two real world Grid workflows we observed that the HEFT algorithm appears to be a good and computationally inexpensive scheduling algorithm that performs better than the other 2 candidates discussed in this paper.

We also investigated a specific class of strongly unbalanced workflows. We demonstrated that any just-in-time scheduling strategy is likely to produce poor results for workflows of this class. Also the workflow partitioning strategy used in Pegasus system [3] appears to have no advantage over the full-graph scheduling, and may produce less efficient results for unbalanced workflows.

We implemented the HEFT algorithm in the ASKALON environment for scheduling scientific workflow applications on the Grid. Future work on the presented scheduling strategy will consist of making it more efficient for heterogeneous environments. We will also examine how typical network scenarios may disrupt the simple scheduling model based on fixed values provided as performance predictions.

8. REFERENCES

[1] P. Blaha, K. Schwarz, G. Madsen, D. Kvasnicka, and J. Luitz. *WIEN2k: An Augmented Plane Wave plus Local Orbitals Program for Calculating Crystal Properties*. Institute of Physical and Theoretical Chemistry, Vienna University of Technology, 2001.

[2] The Austrian Grid Consortium. http://www.austriangrid.at.

[3] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-Hui Su, Karan Vahi, and Miron Livny. Pegasus: Mapping scientific workflows onto the grid. In *European Across Grids Conference*, pages 11–20, 2004.

[4] Peter Rutschmann Dieter Theiner. An inverse modelling approach for the estimation of hydrological model parameters. In *Journal of Hydroinformatics*, 2005.

[5] Rubing Duan, Thomas Fahringer, Radu Prodan, Jun Qin, Alex Villazon, and Marek Wieczorek. Real World Workflow Applications in the Askalon Grid Environment. In *European Grid Conference (EGC 2005)*, Lecture Notes in Computer Science. Springer Verlag, February 2005.

[6] GT 4.0 Security website. http://www.globus.org/toolkit/docs/4.0/security/.

[7] David E. Goldberg. *Genetic Algorithms in Search, Optimization 6 Machine Learning*. Reading. Addison-Wesley, Massachusetts, 1989.

[8] R. L. Graham. Bounds for certain multiprocessing anomalies. In *Bell System Technical Journal 45*, pages 1563–1581, 1969.

[9] Jarek Nabrzyski, Jennifer M. Schopf, and Jan Weglarz. *Grid Resource Management, State of the Art and Future Trends*. Kluwer, 2003.

[10] Radu Prodan and Thomas Fahringer. Dynamic Scheduling of Scientific Workflow Applications on the Grid using a Modular Optimisation Tool: A Case Study. In *20th Symposion of Applied Computing (SAC 2005)*, Santa Fe, New Mexico, USA, March 2005. ACM Press.

[11] Rizos Sakellariou and Henan Zhao. A hybrid heuristic for dag scheduling on heterogeneous systems. In *IPDPS*, 2004.

[12] The Condor Team. Dagman (directed acyclic graph manager). http://www.cs.wisc.edu/condor/dagman/.

[13] Jun Qin Thomas Fahringer and Stefan Hainzer. Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language. In *Proceedings of IEEE International Symposium on Cluster Computing and the Grid 2005 (CCGrid 2005)*, Cardiff, UK, May 9-12 2005. IEEE Computer Society Press.

[14] Veridian Systems. PBS: The Portable Batch System. http://www.openpbs.org.

[15] Henan Zhao and Rizos Sakellariou. An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm. In *Euro-Par*, pages 189–194, 2003.