Problems and Approaches of Workflow Scheduling in MediGRID

Dietmar Sommerfeld
Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen
Göttingen, Germany
Email: dsommer@gwdg.de

Harald Richter
Department of Informatics
Clausthal University of Technology
Clausthal-Zellerfeld, Germany
Email: hri@tu-clausthal.de

***Abstract*—We describe four problems inherent to Grid scheduling that could be identified by means of measurements in the D-Grid. These problems make meta-scheduling nearly always a delicate task. In the face of this, we developed a new hybrid methodology to schedule application workflows which presumably supersedes existing methods. Our algorithm combines existing scheduling strategies for the Grid and for workflows, and it additionally employs three prediction methods for the expected queue waiting times. Three site scenarios could be identified where one respective prediction works best. To meet the dynamic characteristics of heterogeneous Grid resources, we use a list scheduling heuristic to perform full-ahead planning of workflow tasks based on execution time predictions, and then distribute Grid jobs just-in-time according to resource performance predictions calculated from up-to-date monitoring data.**

***Keywords*-Grid, workflow scheduling, queue waiting time prediction, HEFT, just-in-time.**

I. Introduction

In this paper, we investigate the scheduling of Grid workflows within the MediGRID project [1] which is part of the German e-Science initiative D-Grid [2]. MediGRID is a community Grid for researchers in the fields of medicine, biomedical informatics, and life sciences. Four types of pilot applications were selected for the first phase of MediGRID: bioinformatics, image processing, biomedical ontology, and clinical research applications. MediGRID has set up a service Grid that uses the Globus Toolkit 4 (GT4) [3] as its Grid middleware. On top of GT4, MediGRID employs an advanced workflow system for orchestrating the distributed execution of workflow applications on Grid resources. The execution of these applications involves the concurrent and sequential execution of multiple programs, and the automatic and timely data transfer between programs which are also called tasks in workflow terms. A very important issue in executing a scientific workflow application in computational Grids is how to map and schedule workflow tasks onto multiple distributed resources, and how to handle task dependencies.

We describe the problems of scheduling in D-Grid and propose a methodology to schedule workflows that combines existing scheduling strategies for workflow tasks and Grid jobs in a performance-effective approach with manageable complexity. This approach consists of two tiers. In the first tier, a full-ahead schedule of the workflow tasks is created. This means that in a first step the whole workflow is scheduled statically before its execution. For this, we use a list scheduling heuristic similar to the Heterogeneous Earliest-Finish-Time algorithm (HEFT) [4]. In the resulting static schedule, all tasks are assigned ranks which determine static prioritizations. The second tier operates dynamically at runtime and processes the workflow tasks according to their priorities. It performs a just-in-time mapping of tasks to Grid resources and is equivalent to common meta-scheduling on the Grid.

II. Measurements in D-Grid

In the past months, we have conducted a number of experiments to investigate the availability of resources at D-Grid computing sites. A measurement program was installed on five big clusters which are in production use by the MediGRID community. These clusters are located at different D-Grid sites and comprise in total more than 5000 processor cores. All clusters consist of many machines and are controlled by a local resource management system (LRMS) which schedules the jobs in the input queues onto the processors. The measurement program periodically submits test jobs and measures the queue waiting times. As a result of our measurements, we have disclosed four intrinsic problems for meta-scheduling that have to be considered in the future in the design of Grid meta-schedulers.

These problems are:

1) Resources used in Grid computing typically consist of large clusters that are used by several Grid projects and multiple virtual organizations (VO). Each of these VOs can use its own Grid scheduler. In this case, every Grid scheduler only has the role of a "power user" that competes with other users, e.g. other Grid schedulers in the same grid from the perspective of the LRMS. As a consequence, an optimal schedule is not possible for any individual meta-scheduler. We call this the competing-schedulers problem.
2) Computing sites that take part in Grid projects retain the concept of site autonomy. This means that administrative decisions and privileges remain at the site level. Therefore, Grid schedulers have no control over the site scheduler's policy, and over the prioritization of the jobs waiting in the queues. Thus, no control exists over the respective LMRS for

the meta-scheduler. We call this the lack-of-control problem.

3) Every site uses a custom configuration of queues, and processors can be shared among queues or can be dedicated to queues exclusively. Usually the information from the LRMS does not allow to reliably determine the number of available processors. The only statistics commonly available are the number of running and waiting jobs. Some sites do not even provide this information because of nondisclosure agreements. Additionally, sites employ custom configurations for their local scheduling which are usually non-transparent to the users. To conclude, local schedulers currently do not provide sufficient information for a good schedule at the Grid level. We call this the information-insufficiency problem.
4) Resources are normally highly utilized, and waiting times at clusters can last up to hours. Therefore, input queue waiting time considerably exceeds the actual execution time for small jobs and is their dominant factor. However, input queue waiting time and input queue length have shown to be not continuously differentiable functions over time. Instead, they can vary within minutes by a factor of thousand or more. They behave more like fractals than functions. This makes predictions of future queue waiting times and queue lengths a delicate task. However, scheduling always relies on such predictions. We call this the non-continuously differentiable-function problem.

Figs. 1-5 show the variations of waiting time, running and waiting jobs for all five sites during the same measurement interval which was 720 hours (30 days). The left y-axis denotes the waiting time in minutes, the right y-axis the number of waiting and running jobs. The measurement program that collects the data was executed by means of a regular MediGRID user account. This account was used solely for test jobs and did not run production jobs that would consume a notable amount of resources. From site to site, there are a number of typical scheduling criteria: Jobs can be assigned static priorities based on user, group and job class. Fairshare-scheduling prioritizes jobs based on the historical resource usage. Furthermore, jobs can be prioritized depending on the amount of requested resources. The most common method is to raise priority with increasing input queue waiting time. Many sites employ a combination of several criteria. Depending on the configuration of the site scheduler, other users such as non-members of MediGRID can experience greatly different waiting times as our diagrams show.

At site 1, the test jobs almost never had to wait before execution. At the same time, there is no dependency visible between waiting time and the number of waiting jobs, i.e. the test jobs were executed instantly even though other many jobs were already waiting in the queue. The maximum number of running jobs increased during the interval. This means that either the number of processor cores was increased or fewer cores were used by jobs submitted later. Because of the missing correlation between waiting time and waiting jobs, the local scheduling of site 1 probably uses the fairshare method or it assigns very high static priorities to MediGRID jobs.

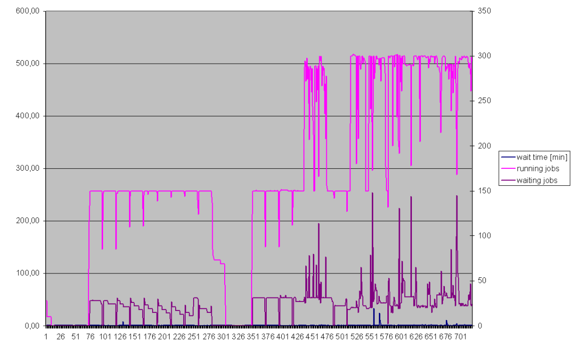

Figure 1. Variation of input queue waiting time and number of jobs in 720 hs at site 1.

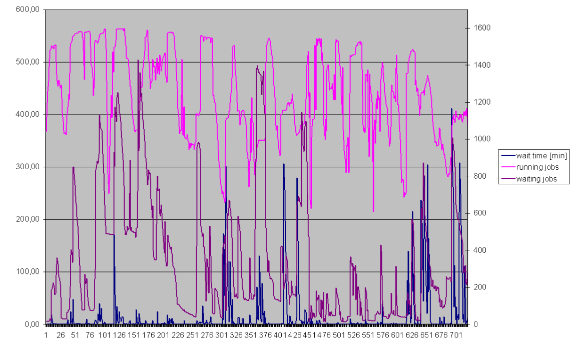

Figure 2. Variation of input queue waiting time and number of jobs in 720 hs at site 2.

The graphs of site 2 show immense variations in queue waiting times. Long periods with very low waiting times are interrupted by steep peaks which last for a few hours. A dependency between waiting times and waiting jobs only seems to exist during the last peaks in the observed interval. This might be an indicator that these jobs received a similar prioritization as the test jobs. The number of running jobs varies a lot while jobs are still queued which points to changes in the parallelism of jobs. The scheduling of site 2 is certainly fairshare-based.

At site 3, the execution of the measurement program was interrupted for a few days, therefore there is a period without data in the center of the 30 day interval. The waiting times show a very clear periodicity of about 24 hours. Additionally, a clear correlation can be observed between waiting time and waiting jobs. The number of running jobs is almost constant. The correlation between waiting time and waiting jobs leads to the assumption that site 3 prioritizes jobs based on the input queue waiting time, i.e. with the first-come first-serve (FCFS) queueing discipline.

During the first seven days of the presented interval,

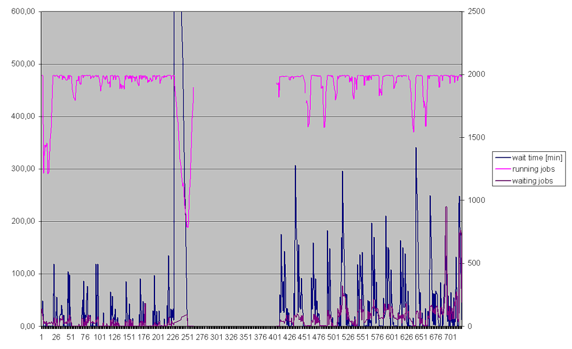

Figure 3. Variation of input queue waiting time and number of jobs in 720 hs at site 3.

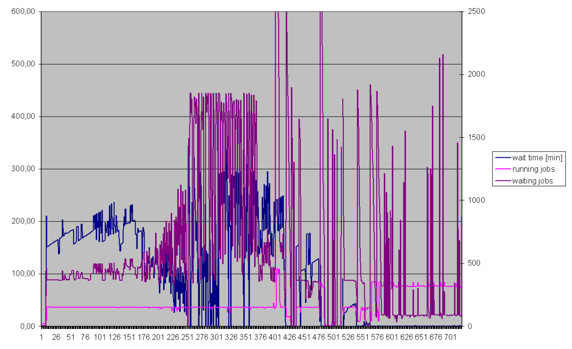

Figure 4. Variation of input queue waiting time and number of jobs in 720 hs at site 4.

waiting times at site 4 lasted on average three hours and correlated with the number of waiting jobs. Afterwards both values show high fluctuation and no consistent correlation. In the last seven days, the test jobs experienced almost no waiting time even though other jobs were waiting in the input queue. The number of running jobs is very constant with a step at the beginning of the last week. The scheduling at site 4 appears to use mainly FCFS, except from the last week. Then, it is changed to fairshare or high prioritization for MediGRID jobs.

The graphs of site 5 show an increasing waiting time, together with an increase in the number of waiting jobs. At the peak, waiting time is almost ten hours, then both values settle down to approximately zero. The second half of the interval is characterized by peaks in waiting time that still correlate with the number of waiting jobs. The number of running jobs remains constant during most of the interval. The scheduling of site 5 is certainly dominated by the FCFS principle.

III. Workflow Management in MediGRID

In MediGRID, the execution of complex workflow applications is handled by the Grid Workflow Execution Service (GWES) [5]. This workflow management system is used in many European Grid projects, e.g. MediGRID, Instant-Grid, K-Wf Grid and CoreGRID [6], [7], [8], [9].

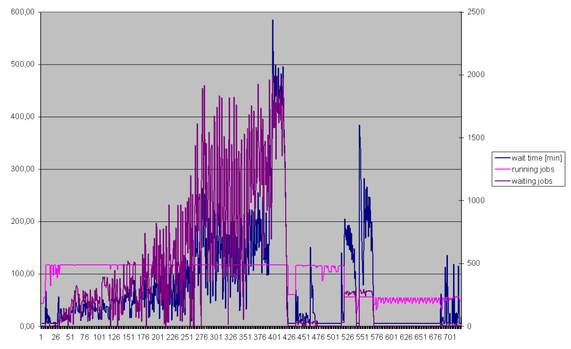

Figure 5. Variation of input queue waiting time and number of jobs in 720 hs at site 5.

The scheduling in MediGRID is accomplished by the scheduler integrated in GWES. When a task is ready to execute, i.e. when all input data are available, GWES automatically selects one suitable machine, based on current utilization. The scheduling algorithm calculates a quality value, i.e. metric, between 0 (busy) and 1 (idle) for all resources. In the end, the scheduler chooses one resource from the sublist of resources whose metric is greater than a given threshold value (e.g. 0.6). This way, the scheduler provides a rough load-balancing between resources and improves job throughput of the Grid.

The current scheduler implementation in MediGRID belongs to the class of just-in-time algorithms. Research shows [10] that just-in-time algorithms can produce good results, however they do not provide any full-graph analysis for task dependencies. Therefore, they lack performance in complex application workflows that have many concurrent tasks. This affects especially the class of unbalanced (asymmetric) workflows with parallel threads that differ significantly in expected thread execution times. In this case, preference has to be given to the longer threads to allow all threads to finish within similar time. The execution of such workflows can be improved by employing a full-graph scheduling as performed by the HEFT algorithm.

IV. Workflow-level Scheduling

A. HEFT

Our new approach is based in its first tier on HEFT [4] which is an extension of the classical list scheduling algorithm based on directed acyclic graphs for heterogeneous environments. HEFT is a simple and computationally inexpensive algorithm, which schedules a workflow by "backward" traversing the directed graph from output to input, constructing an ordered list of tasks, and mapping the tasks to resources.

The HEFT algorithm consists of 3 phases [10]:

1) Weighting: it assigns weights to the nodes and edges in the graph;
2) Ranking: it creates a sorted list of tasks, ordered by their priorities;
3) Mapping: it assigns tasks to resources.

In phase 1, the weights assigned to nodes correspond to the predicted execution times of the tasks, while the edge weights correspond to the predicted data transfer times between the resources. HEFT assumes these times to be known. In environments with homogeneous resources, the weights directly reflect the predicted times. In heterogeneous environments, the weights must be adjusted considering variances in execution times on different resources, and different data transfer times on data links.

In the ranking phase 2, the workflow graph is traversed backward, and a rank value is assigned to each of the tasks. The rank value denotes the task's priority, thus a higher rank means a greater priority. The rank of a task is equal to the tasks' weight plus the maximum successive weight. This means that for every edge leaving the task, the edge weight is added to the previously calculated rank of the successive node, and that the highest sum is chosen. In the end, the tasks are sorted by decreasing rank order. This results in an ordered ranking list.

In the mapping phase, tasks from the ranking list are mapped to the resources one after the other, and each task is assigned to that resource which minimizes the task's earliest expected finish time.

B. Adaptations of HEFT

The value of the predicted execution time a of a task is derived from previous executions of the same task. As the job runtime can vary very much, we use the method of exponential smoothing to estimate the next value based on previous values. The prediction value is calculated as $a_t = \alpha * m' + (1 - \alpha) * a_{t-1}$, where $m' = m * p$. m denotes the latest measured execution time, and a_{t-1} is the previous predicted value of a. p is the performance factor of the resource and used to normalize m with respect to a reference system. p has to be calculated by means of SPEC benchmarks for all Grid resources and is reciprocally proportional to the SPEC value. Even though SPEC benchmarks only allow for a limited classification of compute resources, they are suitable in the MediGRID context, as most jobs are neither I/O-intensive nor parallel. $\alpha \in [0,1]$ is called smoothing factor. We use $\alpha = 0.3$ which is a common choice and gives more weight to the previously predicted values for a stronger smoothing.

The data transfer time b depends on the data size and the bandwidth of the network link connecting the resources. Predicting these values ahead of the workflow execution is difficult because neither data size nor the sites between which the data transfer will take place are known. Additionally, data transfer rates can vary during workflow execution time. Therefore, we assume instead for the workflow-level scheduling an average transfer time of 10 seconds.

Another aspect that has to be considered are the potential queue waiting times of tasks at resources. Many Grid-scheduling research-groups assume a Grid model with high availability and good control over the resources by the scheduler [11], which is often the case for scientific workflows executed in research institutions. However, as our measurements in section II have shown, such assumptions do not hold for the D-Grid environment where availability of resources is limited and where no control exists over the site-level resource-management systems. This is a fundamental constraint that limits the possibilities of meta-scheduling in D-Grid. These circumstances lead to the conclusion that static full-ahead workflow scheduling alone is inappropriate in the D-Grid environment. Therefore, we only employ the first two phases of the HEFT algorithm to calculate static priorities for the tasks. For the mapping phase that assigns tasks to resources, we employ a greatly improved version of the existing dynamic just-in-time scheduling of MediGRID.

V. Grid-level Scheduling

A. Data Locality

The second tier in the proposed approach performs a just-in-time Grid scheduling based on dynamic resource data, short-time predictions and additional information available at runtime. During the workflow processing, tasks that are ready to execute are placed in an internal queue within GWES. In every scheduling cycle, all tasks in the internal queue are rearranged according to their ranks calculated in phase 1 and 2 of the HEFT algorithm and then mapped to the available resources. If sufficient resources are available then no tasks are artificially detained in the internal queue.

As described in section III, the current MediGRID scheduling chooses resources whose quality value is greater than the threshold. We improve resource selection with regard to data locality. Data transfer times can be avoided if tasks are scheduled onto compute resources that already have the required data available. If a data transfer is performed, preference will be given to resources with a fast network connection to the data source. To this end, the data transfer rates between the MediGRID sites are continuously measured using the Grid benchmarking service Jawari [12]. During just-in-time scheduling, source and destination of the transfer as well as data size are known, therefore it is possible to predict transfer time as $t_{transfer} = t_{setup} + size/bandwidth_{(source,dest)}$. After data transfer, the replica has to be registered so that it is available for future job executions.

B. Queue Waiting Time Predictions

Due to differences in queue waiting times, it will often be advantageous to transfer data and to perform the computation on a resource with less queueing delay. However, to correlate transfer and queueing delay, we need to estimate queue waiting time. We propose this method that combines three estimations s, u and v that are based on three different concepts to predict queue waiting time. For all resources, the performance of the three estimations is evaluated and that estimation is used which performed best in the latest scheduling cycles, i.e. the predicted value d_{queue} will be either s, or u or v depending on the accuracy of the previous estimations.

The first estimation, s, works similar to the prediction of job execution time described in the previous section. It uses exponential smoothing to estimate queue waiting time based on previous measurements. The calculation is $s_t = \beta * n + (1 - \beta) * s_{t-1}$, where n denotes the latest measured queue waiting time, and s_{t-1} is the previous estimated value of s. β is the smoothing factor. It is set to $\beta = 0.3$ which has shown by practical experience to sufficiently compensate fluctuations. Multiple values of s should be calculated for different job durations because some sites have exclusive nodes for short and long running jobs or employ backfilling.

The second estimation, u, uses Little's law [13] from queueing theory to calculate the expected value measure of queue waiting time. Application of Little's law requires to determine some basic parameters of the queueing system, e.g. the average rate of jobs entering the queueing system λ. We employ a procedure to measure λ that requires only user permissions. As described in section II, a measurement program was installed on all MediGRID clusters which periodically retrieves the number of running and waiting jobs and the identifier of the latest job. The LRMS uses consecutive numerical values as job identifiers. To reliably determine the identifier of the latest job, the program submits a test job and reads its own job number. The difference between the values of the current and the previous measurements gives the arrival rate λ of jobs in the cluster during a time interval $\lambda = job\_number_t - job\_number_{t-1}$.

According to [13] we define N as the total number of jobs in the system, N_q as the number of jobs waiting in queue, T as the total time a jobs spends in the system, and T_q as the time a job spends waiting in the queue. Then we define the expected value measures of these terms. $L = E[N]$ is the mean number of jobs in the system, $L_q = E[N_q]$ is the mean number in queue, $W = E[T]$ is the mean waiting time in the system, and $W_q = E[T_q]$ is the mean waiting time in queue. Little's laws for general queueing systems (G/G/c) are $L = \lambda W$ and $L_q = \lambda W_q$. Thus, we get the expected queue waiting time as $u = W_q = L_q/\lambda$.

The third estimation, v, is based on a Fourier analysis [14], [15] of the series of measured input-queue waiting-times. We perform a Discrete Fourier transform (DFT) that decomposes the sequence of time values into components of different frequencies. In the frequency domain, the sinusoidal basis functions of the decomposition are analyzed with respect to their amplitudes. Only significant amplitudes are preserved. Amplitudes of low magnitude are set to zero. Afterwards, the frequency domain is transformed back into time domain using the inverse DFT. Since the DFT only generates frequency components needed to reconstruct the finite time segment that was analyzed, its inverse transform cannot reproduce the entire time domain, unless the input happens to be periodic. For this estimation, we assume that the input data is periodic. Thus we obtain the estimated value v as the first value of the periodic extension of the output sequence. The alteration of amplitudes in frequency domain is made to emphasize the periodicity of the sequence.

C. Evaluation of Waiting Time Predictions

To judge the three estimation approaches, calculations were performed with queue data acquired by hourly measurements in the D-Grid. For Little's formula, it turned out to be advantageous to use the current queue length for L_q, and to average λ over the last 12 hours. Figs. 6-10 show exponential smoothing and queueing theory-based estimations in comparison to the actual measured queue waiting times for all five resources during an interval of 288 hours (12 days). An example for DFT data is shown in Figs. 11, 12.

Estimation 1 performs very well for site 1 (see Fig. 6) because the queue waiting times are nearly constant. However, the peaks are predicted one step later than the original which is inherent to first-order filtering. Additionally, after the peaks the prediction fades only slowly. The latter could be improved by choosing a greater smoothing factor, but this would imply more jitter which is not desirable. Estimation 2 performs very bad and is not applicable to site 1.

For site 2 (see Fig. 7), estimation 1 performs reasonable, but suffers even more from the problems described before. Estimation 2 is not able to deliver a useful prediction for site 2.

At site 3 (see Fig. 8), estimation 1 still seems alright at first sight, but the deviation increases with the number of peaks. Estimation 2 performs much better with site 3. Even though it is unable to predict the correct height of the peaks, it does not have a delay as estimation 1. This is especially noticeable at the end of the peaks. The incorrect prediction of the peaks' exact height is a minor issue because as long as the presence of a peak is detected, no jobs are sent to the site anymore.

For site 4 (see Fig. 9), estimation 1 performs reasonable. This site illustrates the importance of the smoothing factor for the suppression of jitter in the queue waiting times. Estimation 2 performs very well in the beginning. In the second half, the magnitude of waiting time is predicted lower than the actual values. However, whenever waiting

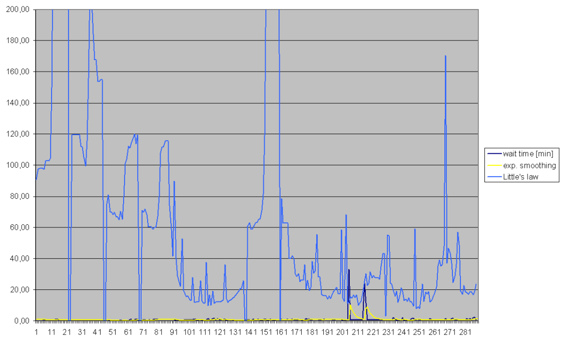

Figure 6. Comparison of measured input-queue waiting-time and estimations for site 1.

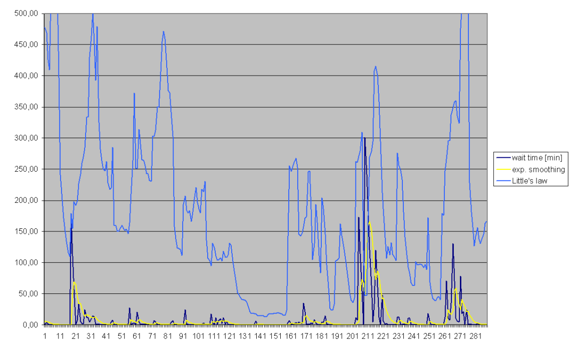

Figure 7. Comparison of measured input-queue waiting-time and estimations for site 2.

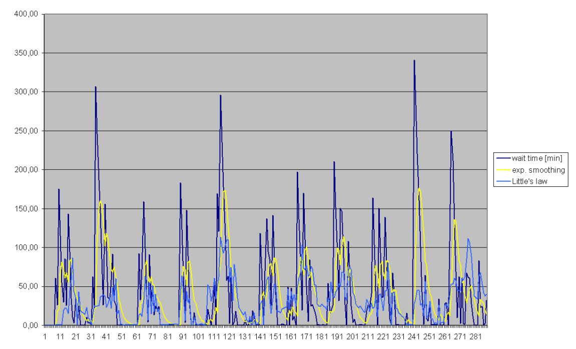

Figure 8. Comparison of measured input-queue waiting-time and estimations for site 3.

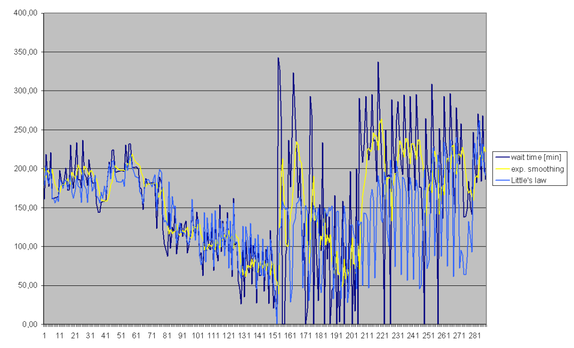

Figure 9. Comparison of measured input-queue waiting-time and estimations for site 4.

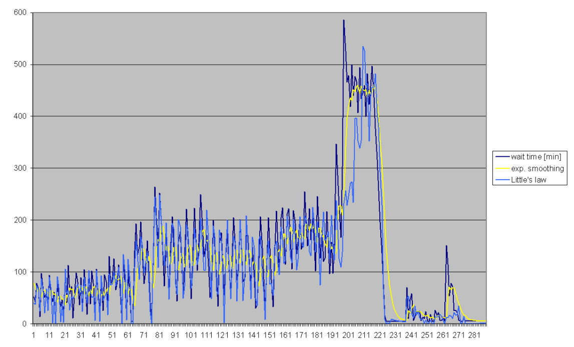

Figure 10. Comparison of measured input-queue waiting-time and estimations for site 5.

time decreases estimation 2 signalizes this earlier than estimation 1.

At site 5 (see Fig. 10), estimation 1 performs similar as with site 4. The main issue is the big delay in descent after the dominant peak in the diagram. Estimation 2 performs very well, only the very last peak is not really predicted.

In Fig. 11, the Fourier spectrum of waiting times of site 3 is shown with original amplitudes as well as with clipped values. The periodicity of the time series data is clearly noticeable from the dominant peak in the frequency spectrum that appears at $f > 0$. The periodic extension shown in Fig. 12 resembles the original continuation very well. It should be pointed out again that only the first yellow point is used from one inverse DFT calculation. The following values are calculated by shifting the input window one step right and repeating the calculation. As can be seen in the graph, the inverse transform can contain negative values, i.e. queue waiting times. Those are clipped to zero.

The depicted measurements and the numerical prognosis showed that all three approaches cover each one scenario best. Queueing performs the better, the more the LRMS configuration matches the first-come first-serve queueing discipline. It becomes unreliable when the arrival rate is very low, or when the actual waiting times are high. Exponential smoothing instead reproduces the trend and the exact value of waiting times. It is the best choice for resources that employ mainly fairshare scheduling or static priorities. However, it suffers from the delay inherent to first-order filtering and needs frequent measurements of the actual queue waiting times. Finally, Fourier analysis avoids the delay of exponential smoothing and is appropriate best for sites that show periodic behavior in queue waiting times.

VI. Related Work

Grid workflow scheduling is still a challenge although the number of approaches is as numerous as the projects dealing with this topic. Related work on the problem is mostly focused on either the domains of workflow scheduling or meta-scheduling. A comprehensive overview of meta-scheduling on the Grid is given in [16].

In [10] Wieczorek compares three scheduling algorithms for scheduling scientific workflows in Grid environments. The scheduling algorithms comprise a genetic algorithm, the HEFT algorithm, and a just-in-time algorithm, similar to the Condor DAGMan resource broker [17]. In a series of experiments, it is shown by Wieczorek that the HEFT algorithm applied with the full-ahead scheduling strategy performs best in practice compared to other approaches.

A comprehensive analysis of multi-criteria grid workflow scheduling is given in [18] and [11]. The authors propose general taxonomies of the problem, based on the

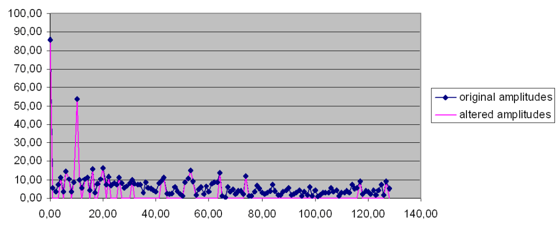

Figure 11. Frequency spectrum for a sequence of waiting times from site 3.

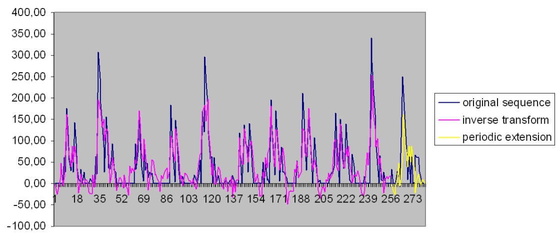

Figure 12. Original sequence of waiting times from site 3 and the inverse transform of altered amplitudes.

aspects (e.g. models, criteria) that influence the decision which scheduling strategy is most appropriate in a given case. Many of the prevalent workflow scheduling approaches for the Grid are analyzed and classified according to the proposed taxonomies.

Recently, the Gridway [19] meta-scheduler has become a popular solution for job scheduling in Globus based Grids. Gridway is part of the GT4 distribution and relies solely on Globus components. The latest version also incorporates support for Condor DAGMan workflows.

VII. Conclusion

Our measurements in D-Grid have disclosed four intrinsic problems that limit the possibilities of meta-scheduling. Since our task was to design a meta-scheduler under these boundary conditions we propose an approach that replaces the current just-in-time Grid scheduling of GWES by an algorithm that makes scheduling decisions based on estimations of execution delay. We found out that three different site scenarios could be identified for which three disjoint estimation methods could be given that predict the respective situation best. Additionally, our approach introduces a preceding workflow-level scheduling-phase that employs parts of HEFT, as well as performance predictions for full-graph analysis. The new approach improves the scheduling performance in case of complex, unbalanced application workflows. Even if the predictions of execution times are incorrect, the performance is expected to be similar to but never worse as the current scheduling method. Future work on the two-tier approach is the practical implementation and test of the proposed scheduling method in MediGRID to show that the new methods work also in a real production environment.

Acknowledgment

This work is partly funded by the German Federal Ministry of Education and Research (BMBF) within the MediGRID project, grant numbers 01AK803A-H.

References

[1] D. Krefting, J. Bart, K. Beronov, O. Dzhimova, J. Falkner, M. Hartung, A. Hoheisel, T. Knoch, T. Lingner, Y. Mohammed, K. Peter, E. Rahm, U. Sax, D. Sommerfeld, T. Steinke, T. Tolxdorff, M. Vossberg, F. Viezens, and A. Weisbecker, "MediGRID: Towards a user friendly secured grid infrastructure," *Future Generation Computer Systems*, vol. 25, no. 3, pp. 326 – 336, 2009.

[2] H. Neuroth, M. Kerzel, and W. Gentzsch, Eds., *German Grid Initiative D-Grid*. Universitätsverlag Göttingen, 2007.

[3] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," in *NPC*, ser. Lecture Notes in Computer Science, H. Jin, D. A. Reed, and W. Jiang, Eds., vol. 3779. Springer, 2005, pp. 2–13.

[4] H. Topcuoglu, S. Hariri, and M. Wu, "Performance-effective and low-complexity task scheduling forheterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.

[5] A. Hoheisel, "Grid Workflow Execution Service - Dynamic and interactive execution and visualization of distributed workflows," in *Proceedings of the Cracow Grid Workshop*, vol. 2, 2006, pp. 13–24.

[6] D. Sommerfeld, T. Lingner, M. Stanke, B. Morgenstern, and H. Richter, "AUGUSTUS at MediGRID: Adaption of a bioinformatics application to grid computing for efficient genome analysis," *Future Generation Computer Systems*, vol. 25, no. 3, pp. 337 – 345, 2009.

[7] C. Boehme, T. Ehlers, J. Engelhardt, A. Félix, O. Haan, T. Kálmán, B. Neumair, U. Schwardmann, and D. Sommerfeld, "Instant-Grid: Fully automated middleware-deployment using a live-CD," in *ICNS '06: Proceedings of the International conference on Networking and Services*. IEEE Computer Society, 2006, p. 70.

[8] "K-Wf Grid," http://www.kwfgrid.eu.

[9] "CoreGRID - Network of Excellence," http://www.coregrid.net.

[10] M. Wieczorek, R. Prodan, and T. Fahringer, "Comparison of workflow scheduling strategies on the grid," *Lecture Notes In Computer Science*, vol. 3911, pp. 792–800, 2006.

[11] M. Wieczorek, A. Hoheisel, and R. Prodan, "Towards a general model of the multi-criteria workflow scheduling on the grid," *Future Generation Computer Systems*, vol. 25, no. 3, pp. 237–256, 2009.

[12] E. de Oliveira, "Jawari - A grid benchmarking service," in *Proceedings of the German e-Science Conference*, Baden-Baden, Germany, 2–4 May 2007.

[13] D. Gross, J. Shortle, J. Thompson, and C. Harris, *Fundamentals of queueing theory*. John Wiley & Sons, 2008.

[14] N. Morrison, *Introduction to Fourier analysis*. Wiley New York, 1994.

[15] T. Butz, *Fourier transformation for pedestrians*. Springer, 2006.

[16] J. Nabrzyski, J. M. Schopf, and J. Weglarz, Eds., *Grid resource management: state of the art and future trends*. Norwell, MA, USA: Kluwer Academic Publishers, 2004.

[17] Condor Team, "DAGMan (Directed Acyclic Graph Manager)," http://www.cs.wisc.edu/condor/dagman/.

[18] M. Wieczorek, A. Hoheisel, and R. Prodan, *Grid middleware and services: challenges and solutions*, ser. CoreGRID. Springer, June 2008, ch. Taxonomies of the Multicriteria Grid Workflow Scheduling Problem, pp. 237–264.

[19] E. Huedo, R. Montero, and I. Llorente, "The GridWay framework for adaptive scheduling and execution on grids," *SCPE*, vol. 2006, p. 2007, 2004.