Efficient Grid Workflow Scheduling Using a Two-Tier Approach

Dietmar SOMMERFELD [a,1], Harald RICHTER [b]
[a] *Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, Germany*
[b] *Department of Informatics, Clausthal University of Technology, Germany*

Abstract.
Research in biomedicine and bioinformatics often requires the analysis of very large data sets. Grid workflows are one means to accelerate this data processing. However, complex workflows comprising of many tasks are not sufficiently supported by state-of-the-art production Grids. We propose and evaluate a two-tier approach that combines contemporary scheduling strategies for Grids and for workflows, and that operates in the harsh environment of production Grids. This combined approach uses the "Heterogeneous Earliest-Finish-Time" algorithm for a full-ahead static schedule of tasks, together with a just-in-time allocation of tasks to resources according to resource performance predictions. For this purpose, sophisticated predictions of input-queue waiting-times and execution times of jobs are utilized. This paper describes our approach, assesses it by measurements and demonstrates thus a 28% acceleration in workflow processing compared to existing strategies.

Keywords. Grid, workflow scheduling, predictions, HEFT, just-in-time.

1. Introduction

Our work was accomplished within the framework of the MediGRID [1] virtual organization which is part of the German D-Grid. MediGRID is a research community for medicine, biomedical informatics, and life sciences. Typical use cases are bioinformatics, image processing, biomedical ontology, and clinical research applications. MediGRID [2] and follow-up projects like Services@MediGRID [3] and PneumoGrid [4] have set up a service Grid that uses the Globus Toolkit 4 (GT4) [5] as Grid middleware. On top of GT4, the Generic Workflow Execution Service (GWES) [6] handles the execution of complex application workflows that consist of several program executions. An important issue in executing biomedical workflows is how to optimally schedule the workflow tasks onto the distributed Healthgrid resources, and how to handle inter-task dependencies to minimize waiting time and to maximize the number of runnable tasks. We present the two-tier approach that combines static list scheduling with dynamic job allocation based on predictions of input-queue waiting and execution times. It has a manageable scheduling complexity and is robust, i.e. suited for production Healthgrids. In the first tier, ranks

[1]Corresponding Author: Dietmar Sommerfeld, GWDG, Am Fassberg 11, 37077 Göttingen, Germany; E-mail: dsommer@gwdg.de

are assigned to workflow tasks according to the “Heterogeneous Earliest-Finish-Time” algorithm (HEFT) [7] in order to establish execution priorities for the tasks. The second tier performs a just-in-time mapping of runnable tasks to Grid resources via an improved MediGRID Grid scheduler that selects one of three methods to optimally predict the queue waiting times of Grid jobs.

The rest of the paper is organized as follows: In section 2, both tiers are described together with the reasons why we employed them. In section 3, the methods and algorithms used for an elaborated input-queue waiting-time prediction are explained. Section 4 presents a metric for balancing performance differences between sites. Section 5 reports about the achieved performance improvements. The paper ends with a conclusion and literature references.

2. Two-Tier Approach

2.1. Tier 1: Workflow-level Scheduling

The first tier of our approach is based on HEFT which is an extension of the classical list scheduling algorithm. It performs a full-graph analysis which is important for unbalanced 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 HEFT algorithm consists of 3 phases [8]. In the weighting phase (1) a task graph is established in which nodes represent tasks, and where directed edges reflect the “depends on” relation. Edges are needed to express inter-task dependencies such as the “read-after-write” condition which means that a successor task needs the output of predecessor tasks as its input in order to become runnable. After the task graph is established, text labels called weights are assigned to all nodes and edges. Each node weight corresponds to a predicted execution time of the respective task, while edge weights denote predicted data transfer times between resources. For heterogeneous site resources, weights must be further processed in order to take into account variances in execution and data transfer times between sites.

The ranking phase (2) traverses the workflow graph from the end nodes back to the starting nodes. Therefore, directed graph edges have to be passed in reversed direction. Phase 2 assigns a rank to nodes. A higher rank means greater priority for the task. The rank of a node is equal to the node’s weight plus the maximum successive weight. This means for every edge leaving the node, that 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 (c.f. [9]). The mapping phase (3) maps tasks from the ranking list to the resources such that each task is assigned to that resource which minimizes the task’s earliest expected finish time.

The value of the predicted execution time of a task is derived from previous executions of the same task. To this end, the runtime of each job is measured during execution. In production use, programs are typically run with similar input data and parameters over a period of time, which results in similar runtimes during that time. Nevertheless, execution times can vary from run to run and from site to site, which is why we smoothed the past measured times with a low-pass filter in order to better predict mean values for the future. The filter uses the method of exponential smoothing which implements a low pass filter of first order. It gives an estimation for the next execution times due to previous values. The prediction value is calculated with $a_t = \alpha * m + (1 - \alpha) * a_{t-1}$. Therein m denotes the latest measured execution time, a_{t-1} is the previously predicted value of a, and $\alpha \in [0,1]$ is called smoothing factor. We use $\alpha = 0.3$ as parameter. This factor was found as optimal by conducting multiple field experiments. It emphasizes previously computed values over new ones for a better smoothing but also reproduces long-term changes in runtimes.

The data transfer time 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 during workflow-level scheduling an average transfer time of 10 seconds.

Another aspect that has to be considered are the potential queue waiting times of jobs at a chosen site resource. Our investigations that were described in [10] have shown that in D-Grid availability of resources is limited and queue waiting times can last up to hours. Additionally, the Grid scheduler has no control over the site-level resource-management systems. These are fundamental constraints that limit the possibilities of meta-scheduling in D-Grid. These circumstances led 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 HEFT to calculate priorities for the workflow tasks. For the mapping phase, we employ an improved version of the existing just-in-time scheduling in GWES.

2.2. Tier 2: Grid-level Scheduling

The second tier of our approach performs a just-in-time Grid scheduling based on predictions and additional information available at runtime. During the workflow processing, tasks that are ready to execute are placed in a queue within GWES. As GWES processes many workflows at the same time, the internal queue holds tasks from several workflows belonging to different users.

While the only optimization goal of the workflow-level scheduling is to decrease the execution time of a single workflow, joint Grid-level scheduling of all tasks allows for Grid-wide improvements. In every scheduling cycle, all tasks in the internal queue are rearranged depending on the prioritization and mapped to the available resources. In the new approach, the prioritization is split up into workflow and task levels.

On the task level, the default operation principle is to sort tasks according to their ranks calculated by the HEFT algorithm. However, tasks of different workflows keep their original order by default. This provides equality between workflows as it prevents workflows that were started later from delaying the earlier ones. That would happen if tasks were sorted regardless of their originating workflows because HEFT assigns the highest ranks to the first tasks within a workflow.

After the prioritization, tasks are submitted by GWES in their final order to the best resources currently available. Ideally, no tasks are retained in the internal queue of GWES if sufficient resources are present. The suitability of resources in terms of security is ensured in MediGRID by the software managers. Only resources where the software has been officially deployed are considered for program executions.

3. Input-Queue Waiting-Time Prediction

3.1. Estimation Methods

For efficient workflow execution, it is mandatory to map runnable tasks to those resources with the smallest expected input-queue waiting-time. Therefore, tier 2 estimates queue waiting times of clusters. We devised three estimation methods called s, u and v that are independent from each other for prediction of queue waiting times.

The first estimation, s uses exponential smoothing according to $s_t = \beta * n + (1 - \beta) * s_{t-1}$, where n denotes the latest measured queue waiting time, s_{t-1} is the previously computed value of s, and β is the smoothing factor. After evaluation by comparison with measured data (c.f. section 3.3) we set $\beta = 0.5$ which results in moderate smoothing and emphasizing of peaks in the series of waiting times because peaks appear frequently.

The second estimation, u, uses Little's law [11] from queueing theory to calculate the expected value of queue waiting time. Little's law requires the average rate of jobs entering the queueing system λ. For this, we employ so-called test jobs that are submitted periodically to all cluster resources. Test jobs are ordinary user jobs and require user permissions only. Let N_q be the number of jobs waiting in a queue, and T_q the time a job spends waiting in the queue. Then we get $L_q = E[N_q]$ and $W_q = E[T_q]$. Little's law states $L_q = \lambda W_q$, thus we get $u = W_q = L_q/\lambda$.

The third estimation, v, is based on a Fourier analysis [12] of the series of measured queue waiting times. We perform a Discrete Fourier transform (DFT) that decomposes the sequence of time values into components of different frequencies. This prediction works only if the input happens to be periodic. This is true for some but not all resources because there are Grid jobs that are submitted automatically at fixed points in time during day and night. In the frequency domain, the sinusoidal basis functions of the decomposition are treated with respect to their amplitudes. All amplitudes of low magnitude are set to zero because they do not contribute much to the signal. Clipping low coefficients results in a filter operation. Afterwards, the remaining coefficients are transformed back into the time domain using the inverse DFT. The estimated value v is the first value of the periodic extension of the resulting output sequence.

Each of the three estimation methods has its pros and cons which is why we developed additionally three selection algorithms that can dynamically choose the presumably best method for a resource's current situation.

3.2. Selection Algorithms

The first selection algorithm uses the Gauss method of least squares and calculates the sum of the squared residuals of prediction and measurement over a fixed time interval. The estimation method that has the minimal squared sum in the considered time interval is chosen for the next prediction. After the prediction was made, all input values for the Gauss evaluation are updated like in a sliding window. The interval was confined by us to the last three measurements, as this gives the best results.

The second selection algorithm incorporates the Gauss method of algorithm 1 and employs additionally a finite-state machine (FSM) with 9 states according to fig. 1. Three states store which of our three estimation methods is currently in use. For each method, an accuracy value is continuously computed by the FSM that comprises the 3 levels

a, b and c, where c is the lowest accuracy. The transitions between the states of the FSM depend on the method of least squares. If the estimation that is currently used has performed best according to Gauss then its accuracy value is increased by one until it reaches the a-grade. Otherwise the accuracy value is decreased as long as it is not lower than c. If the value is already c, the FSM switches to the next estimation method. The three-level grading-scale was chosen because each estimation method should be used at least three times in a row to allow for some steady state.

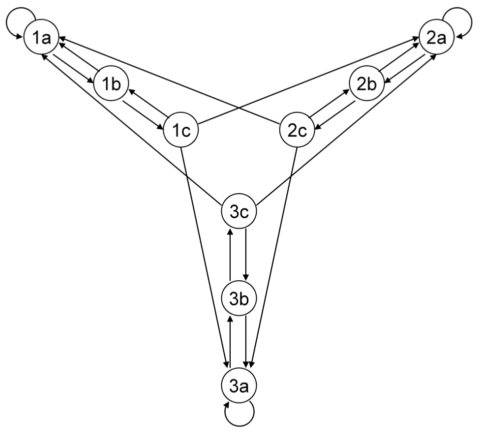

Figure 1. Finite-state machine of selection algorithm 2.

The third selection algorithm is based on three coupled FSMs that are run in parallel (fig. 2). The motivation of this algorithm was to consider each estimation method over a longer time interval while allowing more frequent method changes at the same time. Therefore, each FSM has 6 states named a to f for the accuracy values in order to cover a longer time interval. Dashed edges in fig. 2 depict transitions to other FSMs. To maintain clarity, only the transitions starting from FSM 1 are depicted in fig. 2. The others are analog.

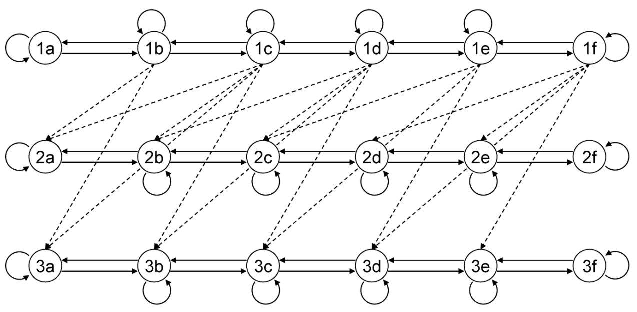

Figure 2. Coupled FSMs for selection algorithm 3.

Each FSM changes its state depending on the accuracy of its estimation method. The six grades a-f correspond to the following deviation between last prediction and measurement, $|d| \leq$ 1.5, 4.5, 13.5, 40, 120, or $>$ 120 minutes, respectively. These deviations have proven most suitable for MediGRID and yielded the best performance among various setups with 4 to 7 states. If the accuracy of the current estimation is better than the FSM state indicates, then the grade is increased by one towards grade a, unless it is already a. Otherwise is decreased by one unless it is f. In each computation cycle, the deviation of each prediction method is updated and algorithm 3 chooses that method with the best accuracy. The actual method is changed when the accuracy becomes worse than a and when one of the two other methods have a better accuracy at the same time.

To summarize, all estimation methods s, u and v are run in parallel on a cluster and the most appropriate one has to be chosen dynamically and automatically by a selection algorithm. To make a decision which of the three developed selection algorithms is most appropriate for MediGRID, all three have been evaluated by measurements. The evaluation was made by comparing predicted with measured job waiting times.

3.3. Evaluation of Selection Algorithms

To investigate the best selection algorithm we have developed a measurement software that periodically submits test jobs to 11 clusters that are used for production service. These resources are located at different D-Grid sites and consist of 500–4000 processor cores each, comprising in total more than 16000 cores. Each cluster is controlled by a local resource management system (LRMS). The LRMS schedules jobs from its local input queues onto the cores. The time interval for the submission of the test jobs was chosen to be 1 hour which is a compromise between disturbance and measurement accuracy. Each measurement job tracks the queue waiting time of the cluster where it was submitted. The collected data was used to evaluate the best selection algorithms. As a result, algorithm 3 was chosen because it turned out to be the best. However, selection algorithms 1 and 2 are not much worse. Table 1 shows the measured performance of the selection algorithms.

With selection algorithm 3, on 7 of the 11 sites nearly 80% of the predictions deviate from reality by not more than 16 minutes. At the 4 other sites, about two thirds of the predictions have the same accuracy. Such deviation values are good, especially if one takes into account that input-queue waiting-times and execution times can last for many hours [10]. This result proves the robustness of our approach, which despite its complexity overall only takes a few seconds to compute.

4. Quality Metrics for Clusters

For the implementation of the two-tier approach we used the latest GWES release 2.1rc9 [13] as starting point. First, the use of waiting time predictions for cluster resources was integrated into this GWES. Grid-level scheduling in GWES is based on a quality metric. The calculation formulas differ for different kinds of resources but all return values in [0,1] to allow a comparison of resources. For clusters the formula is based on the number of running and waiting jobs, $q = e^{-(jobs_{waiting}/jobs_{running})}$.

The idea behind that formula is that a resource with a high computing power always has many running jobs. The absolute computing power of a cluster is translated by the normalization on the number of running jobs into a relative computing power that reflects the potential availability of that cluster. This makes the site's availability comparable to other sites. The formula returns a quality of q=1 if no job are waiting, and it results in a quality of q=0 if the number of running jobs approaches 0.

site	number of	selection	accuracy (%)			
	predictions	algorithm	$\lvert d \rvert \leq 4$ min	$4 < \lvert d \rvert \leq 16$	$16 < \lvert d \rvert \leq 64$	$\lvert d \rvert > 64$ min
		1	93	1	1	3
1	1655	2	93	1	1	3
		3	93	1	1	3
		1	47	19	16	16
2	8784	2	45	19	16	18
		3	49	18	15	16
		1	55	6	8	29
3	1655	2	55	6	8	29
		3	56	5	7	29
		1	85	5	4	4
4	1655	2	84	5	5	3
		3	86	4	4	3
		1	70	16	8	4
5	1537	2	68	17	8	4
		3	71	15	8	4
		1	52	13	17	16
6	1657	2	51	13	17	17
		3	54	12	16	16
		1	75	9	8	6
7	1537	2	72	10	9	6
		3	76	9	7	6
		1	84	3	3	8
8	8784	2	83	3	4	8
		3	84	2	3	8
		1	57	12	15	13
9	8784	2	56	12	15	14
		3	57	11	16	14
		1	85	6	3	4
10	8784	2	83	7	3	4
		3	85	6	3	4
		1	99	0	0	0
11	8784	2	99	0	0	0
		3	99	0	0	0

Table 1. Measured performance of the selection algorithms.

In order to translate waiting times into quality values, we implemented a second function for the quality metric based on a set of definitions. For an input of 1 minute the quality should not be less than 0.9, for 3 minutes approximately 0.8, for 30 minutes approximately 0.4, and for 60 minutes approximately 0.3. The new function is $q' = p^{(\log_2(waittime+1))^r}$, where the parameters p=0.92 and r=1.5 are defined by means of the definitions.

The workflow-level scheduling was implemented in GWES as preceding step of the Grid-level scheduling. It performs the prediction of task execution times which become weights in the first phase of the HEFT algorithm. After all weights have been determined, the ranking phase of HEFT is employed to calculate the priorities of all workflow tasks.

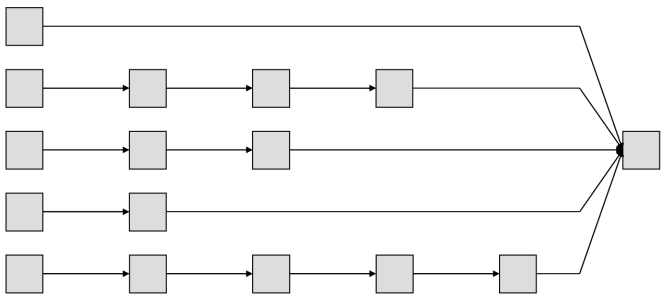

Figure 3. Task graph of the reference workflow.

5. Results

5.1. Test Setup for Prototype

After the implementation of the two-tier approach we assessed its performance by means of a reference workflow. We opted for a synthetic workflow that combines typical characteristics of MediGRID applications. The reference workflow contains sequential and parallel task execution and is depicted in fig. 3. It consists of 16 application parts which are divided into 5 threads of 1 to 5 tasks each that all end in a final joining task. Each of the 15 threaded tasks lasts 60 seconds and is scheduled on the clusters which are in production use by MediGRID. The final task lasts 10 seconds and is bound to one host to collect the results. The task execution times had to be short in order not to disturb productive jobs. Nevertheless, the tests consumed 130 CPU hours on one MediGRID site alone.

For the assessment, we configured a testbed similar to the GWES production environment on the MediGRID portal server. This testbed is based on a typical Grid frontend that in turn uses Globus Toolkit 4. The testbed runs three GWES instances in parallel which are deployed as web applications. The instances are 1) the original GWES 2.1rc9 (denoted as “RC”), 2) an extended GWES with the new Grid-level scheduling (denoted as “TT1”), and 3) a further extended GWES which uses Grid- and workflow-level scheduling (denoted as “TT2”). All three instances processed the reference workflow simultaneously and competed with each other like Grid schedulers from different user communities do. To provide fair conditions, all instances used the same configuration properties as in the production environment.

5.2. Initial Results

In the first test series, we compared RC, TT1, and TT2 in 106 runs of the reference workflow with each other. In each run, the workflow was started at almost the same point in time with only 4 s delays which were needed to prevent the testbed from overload by simultaneous job submissions. The delay represents a slight advantage for RC because its workflows were always started first. At most two test runs were started per hour with a time distance of 30 minutes. The job timeout for the submitted Grid jobs was set to 16 minutes to restrain interference between test runs and to prevent overly long waiting times. The first successful run in every hour was used only for the assessment. Initially, a huge number of unsuccessful workflows happened due to memory problems on the

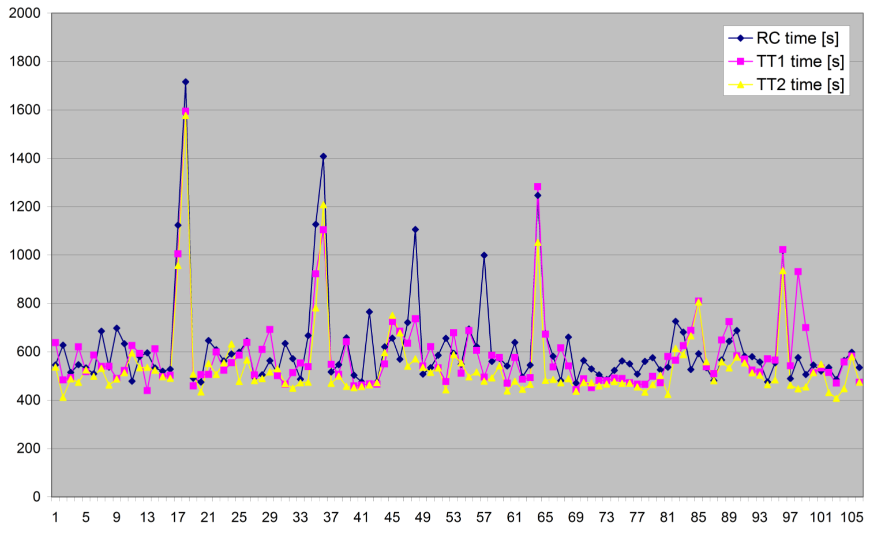

Figure 4. Workflow durations in the first test series.

	RC	TT1	TT2
sum of durations [s]	65282	62498	57226
arithmetic average [s]	615,87	589,61	539,87
standard deviation	194,55	170,91	160,50

Table 2. Evaluation of execution times in series 1.

testbed, bugs in RC, failures of the Grid resources, or job timeouts. Such workflows were not considered for the evaluation. Fig. 4 shows the workflow execution times of all instances in the first test series. The numerical assessment is given in table 2. TT1 has shown a speedup over RC of 1.04, TT2 has a speedup over RC of 1.14. The speedup of TT2 over TT1 is 1.09.

Even though the qualitative improvement is visible in fig. 4, the speedup is small. The reason for this is that most workflows were executed without much waiting time in the input queues because the MediGRID resources were not highly utilized during the first test series due to the absence of a major MediGRID use case. Under such circumstances also the q quality formula performed well. It detected changes quickly because of its high update frequency. However, the waiting time prediction made some false estimations because of its long update interval.

5.3. Advanced Results

In the second test series, we compared TT1 and TT2 in 10 test runs. The aim of this test was to investigate the effect of workflow-level scheduling. For series 2, only the testbed system was used which does not have queue waiting time because tasks are executed as fork jobs. The reference workflow was started by both GWES versions but in different time slots. Only one workflow was running at a time and there was little variance in execution times as fig. 5 shows. The results are given in table 3. The benefit of tier 1 is clearly visible. It delivered a speedup of 1.17 for the given scenario.

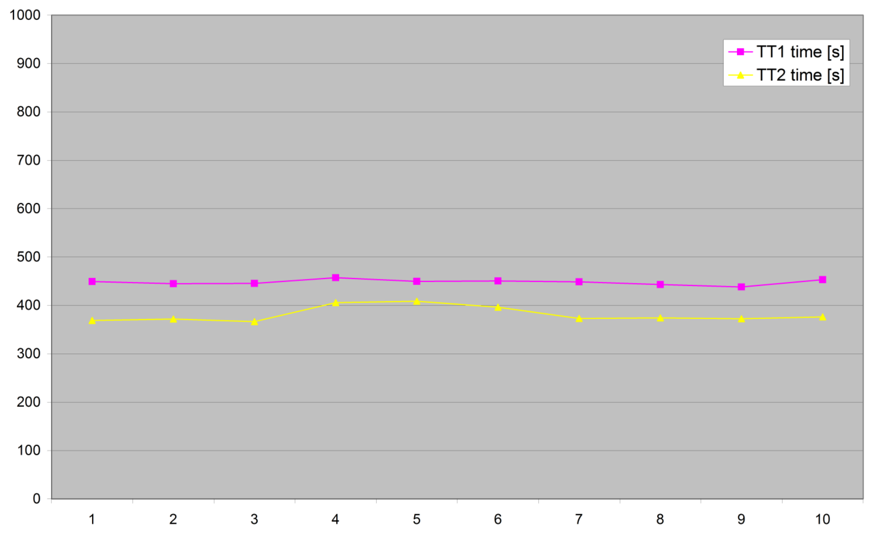

Figure 5. Workflow durations in the second test series.

	TT1	TT2
sum of durations [s]	4480	3813
arithmetic average [s]	448,01	381,31
standard deviation	5,41	15,86

Table 3. Evaluation of execution times in series 2.

5.4. Final Results

An in-depth analysis of test series 1 showed that none of the two formulas for the quality of a cluster were able to react on all peaks in the input-queue waiting-times, i.e. either q or q' remained high although the waiting time was not low. However, most of the peaks were detected either by q or alternatively by q', and the combination of both showed a better hit ratio. Combining both quality metrics can easily be accomplished by multiplying q with q'. The resulting $q''=q \cdot q'$ is again in the interval [0,1] and represents a logical "and" of both ratings. q'' is high only if neither q nor q' detects a peak, and as a consequence, only such resources are selected that have good quality in both quality metrics at the same time.

In test series 3, we compared in 50 test runs RC with a variant of TT1 that uses the combined quality q''. The scenario was identical to test series 1. But in contrast to series 1, workflows were also considered for assessment when they were terminated by job timeouts to include the timeout problem.

Fig. 6 depicts the execution times in a diagram. Workflows with timeout are denoted by red circles. The numerical results are given in table 4. During series 3, much more peaks in the waiting time occurred than in the series 1 because of a higher MediGRID utilization. That preferred RC before TT1 and sometimes led to better results for RC. Nevertheless, series 3 shows a clear improvement for TT1. While RC had timeouts in 16 of 50 test runs, only 2 timeouts happened in TT1. The measured speedup factor of TT1 over RC is 1.28. It would have been even higher without having set timeouts.

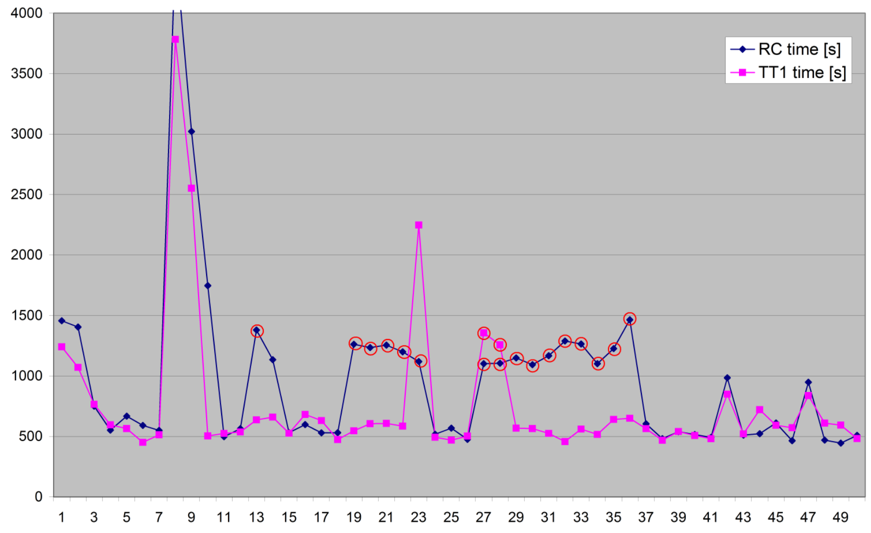

Figure 6. Workflow durations in the third test series.

	RC	TT1
sum of durations [s]	48725	38197
arithmetic average [s]	974,51	763,93
standard deviation	699,72	594,91

Table 4. Evaluation of execution times in series 3.

6. Conclusion

The two-tier approach for the scheduling in MediGRID combines information from previous executions of applications with predictions of future queue waiting times that are retrieved from periodic measurements of input-queues. It replaces the current just-in-time Grid scheduling of MediGRID by an algorithm that makes scheduling decisions based on estimations of execution delay. Additionally, it introduces a preceding workflow-level scheduling-phase that employs parts of HEFT and execution time predictions for full-graph analysis. Thereby, it improves the scheduling performance in case of complex, unbalanced application workflows. At the same time the new approach remains compatible to the current workflow engine, because the second tier keeps the capability to handle all workflow constructs as in the current implementation.

To improve the selection of Grid resources, we developed and evaluated three methods to estimate the queue waiting times of cluster resources. Furthermore, three selection algorithms were described and evaluated to automatically select the best estimation method due to the current situation on the site. The evaluation showed that the selection algorithm that uses 18 states and 6 accuracy levels for every estimation method performs best. With that selection algorithm prediction works very well on most of the examined D-Grid sites and helps to recognize peaks in waiting times which can last up to hours.

Finally, the proposed two-tier approach was assessed by many workflow runs. A testbed was set-up to compare the existing MediGRID scheduler with our two extensions. The results show a performance benefit of 28%. The first extension also decreases runtime in case of no waiting times. The second extension is especially effective in case of high resource utilization. To achieve this, two metrics for the potential availability of cluster resources are employed. By a combination of both metrics, the best results were achieved, resulting in an efficient and very robust scheduler suitable for production Healthgrids. Performance and robustness are indispensable to run biomedical workflow applications in MediGRID. All our modifications have been incorporated in the MediGRID scheduler in GWES. Future work is the deployment of the new scheduler on the MediGRID portal server.

Acknowledgment

This work is supported by the German Federal Ministry of Education and Research within the projects Services@MediGRID (01IG07015) and DGSI (01IG09009) as part of the D-Grid initiative.

References

[1] D. Krefting, J. Bart, K. Beronov, O. Dzhimova, J. Falkner, M. Hartung, A. Hoheisel, T.A. 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*, 25(3):326 – 336, 2009.

[2] 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*, 25(3):337 – 345, 2009.

[3] Services@MediGRID. http://services.medigrid.de.

[4] D. Krefting, S. Canisius, A. Hoheisel, H. Loose, T. Tolxdorff, and T. Penzel. Grid based sleep research – Analysis of polysomnographies using a grid infrastructure. *Future Generation Computer Systems*, In Press, Corrected Proof, 2010.

[5] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. In Hai Jin, Daniel A. Reed, and Wenbin Jiang, editors, *NPC*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13. Springer, 2005.

[6] A. Hoheisel. Grid Workflow Execution Service - Dynamic and interactive execution and visualization of distributed workflows. In *Proceedings of the Cracow Grid Workshop*, volume 2, pages 13–24, 2006.

[7] H. Topcuoglu, S. Hariri, and M. Wu. Performance-effective and low-complexity task scheduling forheterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

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

[9] Dietmar Sommerfeld and Harald Richter. A Two-Tier Approach to Grid Workflow Scheduling. In *2009 2nd IEEE International Conference on Computer Science and its Applications*, pages 267–273, Jeju, Korea (South), 10–12 December 2009.

[10] Dietmar Sommerfeld and Harald Richter. Problems and Approaches of Workflow Scheduling in MediGRID. In *2009 Fifth IEEE International Conference on e-Science*, pages 223–230, Oxford, United Kingdom, 9–11 December 2009.

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

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

[13] Fraunhofer FIRST. Generic Workflow Execution Service. http://www.gridworkflow.org/kwfgrid/gwes-web/.