An Opportunistic Algorithm for Scheduling Workflows on Grids

Luiz Meyer[1], Doug Scheftner[2], Jens Voeckler[2],
Marta Mattoso[1], Mike Wilde[3], Ian Foster[2, 3]

[1]Federal University of Rio de Janeiro - COPPE, Department of Computer Science
[2]University of Chicago - Department of Computer Science
[3]Argonne National Laboratory - Mathematics and Computer Science Division

Abstract. The execution of scientific workflows in Grid environments imposes many challenges due to the dynamic nature of such environments and the characteristics of scientific applications. This work presents an algorithm that dynamically schedules tasks of workflows to Grid sites based on the performance of these sites when running previous jobs from the same workflow. The algorithm captures the dynamic characteristics of Grid environments without the need to probe the remote sites. We evaluated the algorithm running a workflow in the Open Science Grid using tweve sites. The results showed improvement up to 120% relative to other four usual scheduling strategies.

1 Introduction

Grids [9] are emerging as virtual platforms for high performance and integration of networked resources. In these environments, distributed and heterogeneous resources owned by independent organizations can be shared and aggregated to form a virtual computer. Scientific applications usually consist of numerous jobs that process and generate large datasets. Frequently, these components are combined generating complex scientific workflows. Therefore, scientific communities like physicists, biologists, astronomers are using the grid computing to solve their complex large-scale problems.

Processing scientific workflows in a Grid imposes many challenges due to the large number of jobs, file transfers and the storage needed to process them. The scheduling of a workflow focuses on mapping and managing the execution of tasks on shared resources that are not directly under the control of the workflow systems [23]. Thus, choosing the best strategy for a workflow execution in a Grid is a challenging research area.

Often, a scientific workflow can be represented as a Directed Acyclic Graph (DAG) where the vertices represent tasks and the edges represent data dependencies. One alternative to process this kind of workflow is to statically pre-assign tasks to resources based on the information of the entire workflow. This strategy can be used

Candidate to the Best Student Paper Award

by a planner to optimize the execution plan for the DAG [6]. However, since a Grid execution environment can be very dynamic, this alternative may produce poor schedules because by the time the task is ready to run the resource may be unavailable. Besides, it is not easy to accurately predict the execution time of all tasks. Another scheduling approach is to perform the assignment of tasks to resources dynamically as soon as the task is ready to be executed. In this case, if a resource is not available, it will not be selected to process the task. However, many sites can be available to run the task, and selecting the best one can be done according to many alternatives, like number of processors in the site, load balance or data availability.

This work presents an algorithm, which we name *Opportunistic*, which dynamically assigns jobs to grid sites. The algorithm adopts an observational approach and exploits the idea of scheduling a job to a site that will probably run it faster. The opportunistic algorithm takes into account the dynamic characteristics of Grid environments without the need to probe the remote sites. We compared the performance of the opportunistic algorithm with different scheduling algorithms in a context of a workflow execution running in a real Grid environment. We conducted our experiments using the Virtual Data System (VDS) [10], which defines an architecture to integrate data, programs, and the computations performed to produce data. VDS combines a virtual data catalog for representing data derivation procedures and derived data with a virtual data language that enables the definition of the workflows. VDS also provides users with two planners that schedule jobs onto the grid and manage their execution. Scheduling in VDS can be done according to a family of site selectors available for user needs. This work extends the library of site selectors with a new *Opportunistic* site selector algorithm. Our results with experiments in a real Grid environment suggest that the opportunistic algorithm can increase performance up to 120% when compared to scheduling algorithms, currently adopted in most systems, particularly in VDS.

The rest of this paper is organized as follows. Section 2 discusses the related work that deals with grid scheduling. Section 3 describes the Virtual Data System architecture where the opportunistic strategy was implemented while in section 4, we detail the opportunistic algorithm. In section 5 we describe the experiments performed and in section 6 the experimental results are analyzed. Finally, section 7 concludes this work and points to future directions.

2 Related Work

Finding a single best solution for mapping workflows onto Grid resources for all workflow applications is difficult since applications and Grid environments can have different characteristics [23]. In general, scheduling workflow applications in distributed environments is done by the adoption of heuristics. There are many works in the literature addressing the benefits of the scheduling based on data locality in scenarios of data grids. Casanova et al. [2] propose an adaptive scheduling algorithm for parameter sweep applications where shared files are pre-staged strategically to improve reuse. Ranganathan and Foster [17, 18] evaluate a set of scheduling and replication algorithms and the impact of network bandwidth, user access patterns and

data placement in the performance of job executions. The evaluation was done in a simulation environment and the results showed that scheduling jobs to locations where the input data already exists, and asynchronously replicating popular data files across the Grid provides good results. Cameron et al. [4, 5] also measure the effects of various job scheduling and data replication strategies in a simulation environment. Their results show benefits of scheduling taking into account also the workload in the computing resources. Mohamed and Epema [14] propose an algorithm to place jobs on clusters close to the site where the input files reside. The algorithm assumes knowledge about the number of idle processors and the size of the input file for scheduling a job.

The workloads studied in these works consist of a set of independent jobs submitted from different users spread over different sites. Our work differs by focusing on scheduling jobs belonging to a single application, which is a workflow, submitted from a single user in a single site.

Many researchers have studied scheduling strategies for mapping application workflows onto the grid. Ammar et al. [1] developed a framework to schedule a DAG in a Grid environment that makes use of advance reservation of resources and also considers the availability knowledge about task execution time, transfer rates, and available processors to generate a schedule. Their simulation results show advantages of unified scheduling of tasks rather than scheduling each task separately. Mandal et al. [13] apply in-advance static scheduling to ensure that the key computational steps are executed on the right resources and large scale data movement is minimized. They use performance estimators to schedule workflow applications. Wieczoreket et al. [22] compare full graph scheduling and just-in-time strategies for scheduling of scientific workflow in a Grid environment with high availability rate and good control over the resources by the scheduler. Their results show best performance for full graph scheduling. Deelman et al. [6, 7] can map the entire workflow to resources at once or portions of it. This mapping can be done before or during the workflow execution. Their algorithm prefers to schedule computation where data already exist. Additionally, users are able to specify their own scheduling algorithm or to choose between a random and a round robin schedule technique. Dumitrescu et al. [8] studied the performance execution of Blast jobs in Grid3 [11] according to several scheduling algorithms. In their experiments they used a framework that considered resource usage policies for scheduling the jobs. Their results showed that random and round-robin algorithms achieved the best performance for medium and large workloads.

Triana [12] allows scientists to specify their workflows which can be scheduled directely by the user or by the GriLab Resource Management System. In this case, the scheduling is done according to requirements specified for each task. Taverna [15] provides a set of tools to define bioinformatics workflows based on a composition of web services, but not much detail is given about the scheduling of tasks.

In our work, we also deal with the problem of scheduling jobs belonging to a single application, which is a workflow expressed as a DAG. Like in the previous workflow scheduling works, the goal of the scheduling is to minimize the overall job completion. In our algorithm, the planning scheme is completely dynamic and based on an observational approach. We do not consider performance estimation of Grid resources, use of advance reservations or requirements specifications. The

performance evaluation was conducted in a real Grid environment without any control over the resources.

3 VDS planning architecture

In VDS, users specify their workflows through the use of VDL [10]. The VDC (Virtual Data Catalog) stores the user's workflow definition and provides the planner with the logical file names of the files and the name of the transformations (executable programs). The Replica Catalog provides the physical name for the input files given their logical file names. The transformation catalog (TC) specifies how to invoke (executable name, location, arguments) each program. Finally, the Pool Configuration catalog is responsible to provide the information about the desirable grid sites to run the workflow. Figure 1 illustrates the VDS planner architecture.

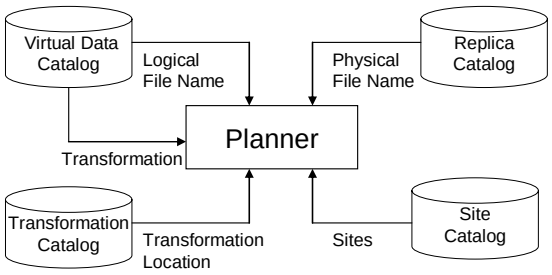

Fig. 1. VDS Planner architecture

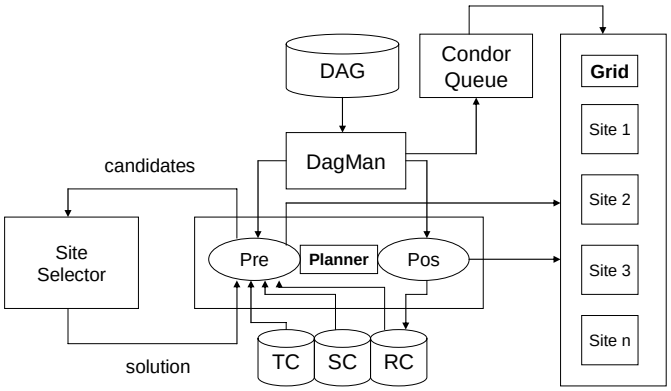

Fig. 2. VDS Planning mechanism

The planner makes use of a site selector mechanism in order to schedule each job of the workflow. The goal of the site selector is to choose a grid site capable to execute a given job. In the VDS planner, the pre-script dynamically builds a list of the available sites for executing each job based on the information of the Transformation Catalog and Pool Configuration. The Pre-script then calls the site selector mechanism and waits for the solution, that is, the site selected for the job execution. The solution returned by the site selector is passed to Dagman [3], which is responsible for scheduling the job into the grid.

Figure 2 details the planner's functionality and its interaction with a site selector. After receiving the identification of the site to run the job, the VDS-Planner executes the replica selection by querying the Replica Location Service to locate all replicas for each file. If there is a replica located in the selected site then this replica will be chosen. Otherwise, the planner will perform a third party transfer of the input files from the sites where they are located to the site where the job is supposed to run. Whenever a job ends, all input files dynamically transferred for the job execution site are erased in the post-processing step.

4 The opportunistic algorithm

Scheduling workflow tasks in Grid environments is difficult because resource availability often changes during workflow execution. The main idea of the opportunistic algorithm is to take advantage of this environment changes without needing to probe the remote sites. In order to implement our opportunistic algorithm using VDS, a few extensions were promoted in the system: we created a control database for logging the location and the status of the workflow jobs, and coded a site selector routine responsible for choosing the execution site for a job. Since the control database is updated by the postscript of each job, the VDS postscript code also had to be modified.

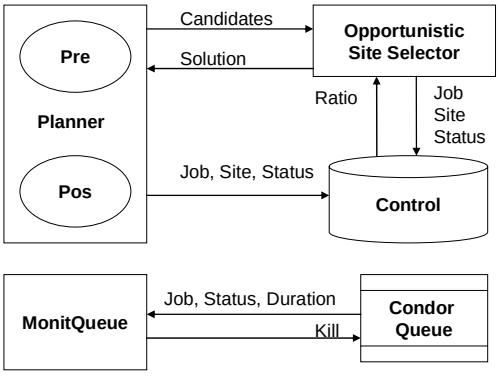

Fig. 3. The opportunistic site selector architecture

The goal of the opportunistic site selector is to select a site to run a job based on the performance of each site when running previously jobs of the same workflow. In

other words, the site selector assigns more jobs to sites that are performing better, according to the architecture in Figure 3. The performance is measured by the ratio (number of ended jobs / number of submitted jobs) at each site, as shown in the algorithm from Figure 4. As long as no jobs have completed, the site selector performs a round robin distribution between the sites. In order to keep track of the submissions and completions of the workflow jobs, the site selector makes use of a control database.

Algorithm: *Opportunistic*
Input: Job *J* to be submitted.
Set S_e $\{s_i\}$ of availables sites informed by the planner.
Set S_o $\{s_i\}$ of sites informed by the *Control Database*
$f_1(S) \rightarrow$ Number of jobs scheduled to site S
$f_2(S) \rightarrow$ Number of jobs ended at site S
$f_3(max) \rightarrow$ Site;
$f_4(min) \rightarrow$ Site;
$f_5 \rightarrow \{j_i\}$; Set of submited jobs
Output: *Solution* - Identification of the Site selected to run the job.
Initialization:
$flag \leftarrow 0$
$min \leftarrow$ *high value*
$max \leftarrow$ *low value*
Steps:
1. **Foreach** site $s_i \in S_o$ **do**
  1.1 **if** $s_i \in S_e$ **then**
    1.1.1 $T_{i,s} \leftarrow f_1(s_i)$
    1.1.2 **if** $T_{i,s} < min$ **then**
      1.1.2.1 $min \leftarrow T_{i,s}$
    1.1.3 $T_{i,c} \leftarrow f_2(s_i)$
  1.2 **if** $T_{i,c} > 0$ **then**
    1.2.1 $R_i \leftarrow (T_{i,c} / T_{i,s})$
    1.2.2 $flag \leftarrow 1$
    1.2.3 **if** $R_i > max$ **then**
      1.2.3.1 $max \leftarrow R_i$
2. **if** $flag = 1$ **then**
2.1 $Solution \leftarrow f_3(max)$
2.2 **else** $Solution \leftarrow f_4(min)$
3. $T_o \leftarrow f_5$
4. **if** $T \in T_o$ **then**
  4.1 **update** *siteid* for job *T*
  4.2 **else insert** tuple (*T*, *solution*)
5. **Return** *Solution*

Fig. 4. The opportunistic algorithm

Whenever the site selector chooses a site to run a job, one record is inserted in the control database with the identifications of the job, the identification of the selected site and a status set to “submitted.” Whenever a job ends, another record with job identifier, site, and status set to "ended" is also added. This last insertion is done by the postscript of every job.

The second component of the opportunistic approach is a queue monitor for the submitted jobs of the workflow. The main motivation to develop this component is that is very usual to have jobs submitted waiting for execution in remote queues. The goal of *MonitQueue* is to keep track of the jobs submitted by DagMan/Condor in order to remove those jobs that are not presenting a desired performance. In the actual implementation, the user must inform the maximum time a job can wait in a queue in an idle status. When a job reaches this time it is killed and automatically re-planned by Euryale. In this case, the opportunistic site selector will choose another site to run the job.

5 Experiments

Many scientific applications can be characterized as having sets of input and derived data that have to be processed in several steps by a set of programs. These batch-pipelined workloads [21] are composed of several independent pipelines and each pipeline contains sequential processes that communicate with the preceding and succeeding processes via data files.

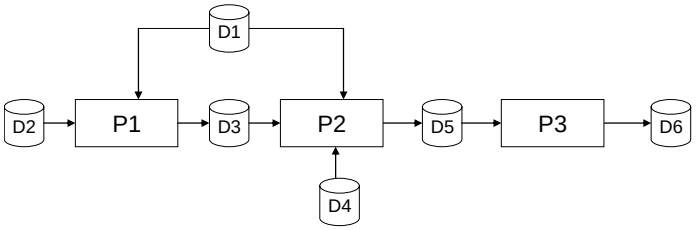

Fig. 5. The pipelined workflow

We defined a pipelined workflow to evaluate a set of scheduling strategies in this experiment. The design of the workflow is shown in figure 5 while figure 6 depicts the corresponding DAG. There is an input dataset D1 with only one file which is input for the first and second programs in the pipeline. The first program also has as input file, a file belonging to dataset D2. Program P2 process the output generated by P1 and also has more two files as input for its processing: the file from D1 dataset and a file from dataset D4. The third and last program of the pipeline processes the output file produced by P2 and outputs a file for the dataset D6. The width of the pipeline was set to 100 nodes in each level, totalizing 300 jobs in the workflow.

Currently, the VDS system provides three choices for the planners: *Round-Robin*, *Random* and *Weighted-Linear-Random*. We evaluated the *Opportunistic* algorithm against the *Weighted-Linear-Random*, *Round-Robin, Last-Recent-Used* and *Data-*

Present algorithms. The last two strategies were coded for the experiments. The overall ideas of these algorithms are:

1. *Weighted-Linear-Random (WLR)*- The execution site is selected randomly but sites with more processors receive more jobs to process.
2. *Round-Robin* - Jobs are sent to sites in a round-robin way. Thus, the number of jobs assigned to each site is the same.
3. *Last-Recent-Used(LRU)* - The execution site corresponds to the site where the last job ended.
4. *Data-Present* - A job is sent to a site with the most files that it needs. If more than one site qualifies then a random one is chosen.
5. *Opportunistic* - The execution site is selected according to the performance of each site. This performance is measured by dividing the number of concluded jobs by the number of submitted jobs at each site. While there are no jobs concluded, a round-robin scheduling is performed.

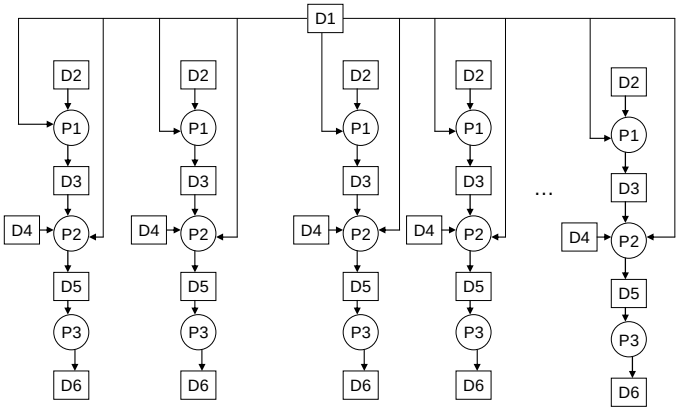

Fig. 6. The DAG shape for the pipelined workflow

Table 1 - Resources available by OSG sites

Site	Processors
UIOWA_OSG_PROD	6
HAMPTONU	8
PURDUE_PHYSICS	63
UFLORIDA_IHEPA	70
UWMADISON	83
UC_ATLAS	110
UTA_DPCC	148
UERJ_HEPGRID	160
CIT_T2	224
UWMILWAUKEE	304
OSG_LIGO_PSU	314
USCMS_FNAL	989

We conducted the experiments using twelve sites from the Open Science Grid [16]. Table 1 shows a snapshot of the total resources available at each site. In order to not interfere with the production, we defined all workflow jobs as sleep jobs. We used two different machines at University of Chicago for running Dagman and the replica catalog respectively. A third machine at the same site was used to store all the input files for the workflow. Table 2 shows the average execution time and transfer time for each type of job and file of the workflow. The size of all input and output files is one megabyte.

Table 2 - Average time in seconds for data transfer and execution according to the type of the workflow level

	P1	P2	P3
Number	100	100	100
Transfer time	17	27	13
Execution time	300	120	60

6 Results

We executed the workflow twenty times for each scheduling strategy, totalizing 30,000 job executions. Figure 7 presents the performance results for all algorithms. As can be noted, the performance of the five algorithms is almost the same during the execution of the first hundred jobs of the workflow. *Opportunistic*, *Last-Recent-Used* and *Round-Robin* algorithms adopt the same scheduling strategy while there is no job concluded. The *Data-Present* algorithm uses a strategy similar to *Weighted-Linear-Random* while there is no site with the needed input files for the job. Since there is no dependencies among the jobs in the first level of the workflow, Condor/DagMan can submit them as soon as the pre-script of each job is finished. The time to transfer the input files is very low and consequently the pre-processing for each job is very fast causing most jobs in this level to be scheduled before any job had finished. As soon as the jobs in the first and second levels begin to finish, *Opportunistic*, *Last-Recent-Used* and *Data-Present* start to schedule according different approaches. *Opportunistic* and *Last-Recent-Used* use their observational characteristics while *Data-Present* takes advantage of data locality. In the first case, the scheduling starts to be done based on the ratio (*jobs concluded/jobs submitted*) while *Last-Recent-Used* starts to schedule to the site that finished the processing of the last job.

The *Opportunistic* algorithm benefits from the dynamic aspects of the Grid environment. If a site happens to perform poorly, then the number of jobs assigned to this site decreases. Similarly, if a site process jobs quickly, then more jobs are scheduled to that site.

The *Last-Recent-Used* algorithm may not present a good performance when a job is scheduled to a site and have to wait a long period of time in the remote queue. When this happens, the next job in the workflow will probably show the same performance problem because it must be scheduled to run in the same site. This kind

of problem is avoided by the *Opportunistic* algorithm because a job can be cancelled by *Moniqueue* if it was not started after a determined period of time.

Since the size of the files generated during the execution is small, the time to transfer these files does not impact the performance and does not bring benefits to *Data-Present* algorithm.

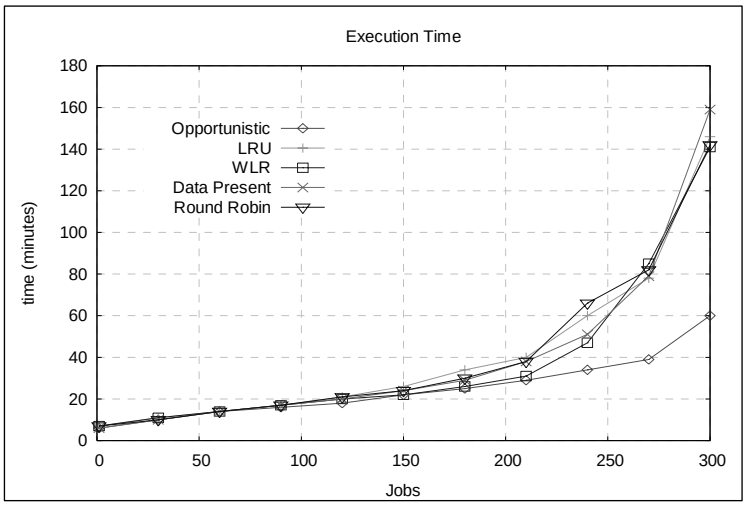

Fig. 7. Execution time by algorithm

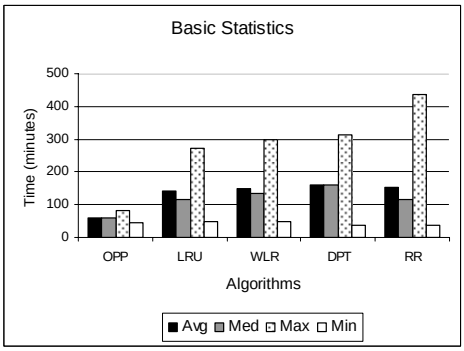

Fig. 8. Basic execution statistics

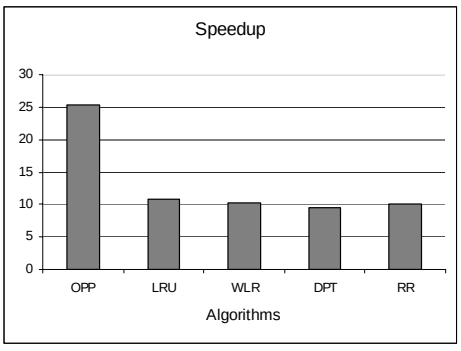

Fig. 9. Speedup by algorithms

Round-Robin provides a good load balance among the sites but since the performance varies among sites, scheduling the same number of jobs to each site is not beneficial. *Weighted-Linear-Random* does not show a good performance because scheduling more jobs to sites with more resources does not guarantee better results since jobs can have to wait in the remote queues. It seems that this kind of strategy is more indicated to Grid environment where resources can be reserved for the entire execution of the workflow.

Figure 8 shows a set of few basic statistics about the workflow runnings. As can be observed, the minimum execution time is almost the same for all algorithms. This occurs because eventually all sites may be presenting a good performance due to having processing resources available by the time of the execution. However, the most usual behavior is to have sites presenting different performance as the workflow is being processed. Consequently, the median, average and maximum execution time differ according to the execution strategy.

Figure 9 shows the speedup of the five algorithms. The execution of the workflow with the opportunistic algorithm was approximately twenty five times faster than running in a single machine. The speedup achieved by the Opportunistic algorithm was more than 150% higher than the other strategies.

7 Conclusions and Future Work

We have proposed a new "opportunistic" algorithm for scheduling jobs in grid environments, and compared its performance with other algorithms. In particular, we analyzed the performance with a very common workflow pattern, a pipeline of programs. The results showed that the *Opportunistic* algorithm provided superior performance when compared to other four VDS algorithms for scheduling workflows jobs. The performance improvement is achieved as a consequence of the observational approach implemented by the algorithm. This approach exploits the idea of scheduling jobs for sites that are presenting good response times and to cancel jobs that are not being executed after a period of time. The algorithm is not aware of sites capabilities and does not need to collect data from remote sites being easy to implement and can be used by other workflow engines.

We intend to perform more comparative experiments with other scheduling algorithms to confirm the efficiency of the *Opportunistic* algorithm. We also intend to study the performance of the algorithm when dealing with other workflow patterns and sizes, and to promote extensions in order to analyze the impact of dealing with different sizes of historical data to compute a site's value.

Acknowledgements

This work is supported in part by the National Science Foundation GriPhyN project under contract ITR-086044, U.S. Department of Energy under contract W-31-109-ENG-38 and CAPES and CNPq Brazilian funding agencies.

References

1. Ammar H. Alhusaini, Viktor K. Prasanna, C.S. Raghavendra. "A Unified Resource Scheduling Framework for Heterogeneous Computing Environments," *hcw*, p. 156, Eighth Heterogeneous Computing Workshop, 1999.

2. Casanova, H., Obertelli, G., Berman, F., Wolski. R., The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid, in SuperComputing 2000, Denver, USA, 2000.
3. DagMan, http://www.cs.wisc.edu/condor/dagman/.
4. D.G. Cameron, R. Carvajal-Schiaffino, A.P.Millar, Nicholson C., Stockinger K., Zini, F., Evaluating Scheduling and Replica Optimisation Strategies in OptorSim, in Proc. of 4th International Workshop on Grid Computing (Grid2003). Phoenix, USA, November 2003.
5. D.G. Cameron, R. Carvajal-Schiaffino, A.P.Millar, Nicholson C., Stockinger K., Zini, F., Evaluation of an Economic-Based File Replication Strategy for a Data Grid, in Int. Workshop on Agent Based Cluster and Grid Computing at Int. Symposium on Cluster Computing and the Grid (CCGrid2003), Tokyo, Japan, May 2003.
6. Deelman,E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M., Vahi, K., Livny, M., Across Grids Conference 2004, Nicosia, Cyprus
7. Deelman, E., Blythe, J., Gil, Y., Kesselman,C., Workflow Management in GriPhyn, The Grid Resource Management, Netherlands 2003.
8. Dumitrescu, C, Foster, I., Experiences in Running Workloads over Grid3, GCC 2005, LNCS 3795, pp.274-286, 2005.
9. Foster, I., Kesselman, C., 1999, Chapter 4 of "The Grid 2: Blueprint for a New Computing Infrastructure", Morgan-Kaufman, 2004.
10. Foster, I., Voeckler,J., Wilde,M., Zhao, Y., Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation, in 14th International Conference on Scientific and Statistical Database Management (SSDBM 2002), Edinburgh, July 2002.
11. Foster, I. et al.,The Grid2003 Production Grid: Principles and Practice, in 13th International Symposium on High Performance Distributed Computing, 2004.
12. GOODALE, T., TAYLOR, I., WANG, I., "Integrating Cactus Simulations within Triana Workflows", In: Proceedings of 13th Annual Mardi Gras Conference - Frontiers of Grid Applications and Technologies, Louisiana State University, pp. 47-53, February, 2005.
13. Mandal, A., Kennedy, K., Koelbel, C., Marin, G., Crummey, J., Liu, B., Johnsson, L., Scheduling Strategies for Mapping Application Workflows onto the Grid, The 14th IEEE International Symposium on High-Performance Distributed Computing (HPDC-14), Research Triangle Park, NC, USA, July 2005.
14. Mohamed, H.H., Epema, D.H.J., An Evaluation of the Close-to-Files Processor and Data Co-Allocation Policy in Multiclusters, IEEE International Conference on Cluster Computing, San Diego, USA, September 2004.
15. Oinn, T., ADDIS, M., FERRIS, J. et al, 2004, "*Taverna: a Tool for the Composition and Enactment of Bioinformatis Workflow*", In: *BIOINFORMATICS*, vol. 20, no 17 2004, pp. 3045-3054, Oxford University Press.
16. Open Science Grid, http://www.opensciencegrid.org
17. Ranganathan,K., Foster,I., Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids, in Journal of Grid Computing, V1(1) 2003.
18. Ranganathan,K., Foster,I., Computation Scheduling and Data Replication Algorithms for Data Grids, 'Grid Resource Management: State of the Art and Future Trends', J. Nabrzyski, J. Schopf, and J. Weglarz, eds. Kluwer Academic Publishers, 2003.
19. Shan,H., Oliker, L., Smith, W., Biswas, R., Scheduling in Heterogeneous Grid Environments: The Effects of Data Migration, International Conference on Advanced Computing and Communication, Gujarat, India, 2004.
20. Singh, G., Kesselman, C., Deelman, E., Optimizing Grid-Based Workflow Execution, work submitted to 14th IEEE International Symposium on High Performance Distributing Computing, July 2005.

21. Thain,D., Bent,J., Arpaci-Dusseau, A., Arpaci-Dusseau,R., Livny, M., Pipeline and Batch Sharing in Grid Workloads, 12th Symposium on High Performance Distributing Computing, Seattle, June 2003.
22. Wieczorek, M., Prodan, R.,Fahringer,T., Scheduling of Scientific Workflows in the ASKALON Grid Environment, SIGMOD Record, Vol. 34, No.3, September 2005.
23. Yu,J., Buyya, R., A Taxonomy of Scientific Workflow Systems for Grid Computing, SIGMOD Record, Vol.34, No.3, September 2005.
24. Zhang, X., Schopf, J., Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, Proceedings of the International Workshop on Middleware Performance (MP 2004), part of the 23rd International Performance Computing and Communications Conference (IPCCC), April 2004.