Performance Analysis of Grid DAG Scheduling Algorithms using MONARC Simulation Tool

Florin Pop*, Ciprian Dobre*, Valentin Cristea*
*Faculty of Automatics and Computer Science, University Politehnica of Bucharest, Romania
E-mails: {florinpop, cipsm, valentin}@cs.pub.ro

Abstract

This paper presents a new approach for analyzing the performance of grid scheduling algorithms for tasks with dependencies. Finding the optimal procedures for DAG scheduling in Grid systems is important especially in large scale distributed computing systems and complex applications for different research areas. We propose to evaluate DAG scheduling algorithms using simulation, an approach suitable for different scheduling algorithms using various task dependencies and considering a wide range of Grid system architectures. Our proposed solution is based on MONARC, a generic simulation framework designed for modeling large scale distributed systems. We present our work in extending the simulation platform to accommodate various DAG scheduling procedures and, as a case study, we present a critical analysis of four well known DAG scheduling strategies: CCF (Cluster ready Children First), ETF (Earliest Time First), HLFET (Highest Level First with Estimated Times) and Hybrid Remapper. The obtained results confirm that the proposed solution is a very good vehicle for performance evaluation in a wide range of DAG scheduling algorithms and a large scale distributed system architectures.

Keywords: Grid Scheduling, DAG Scheduling, Simulation, MONARC, Performance Analysis.

1. Introduction

The actual increasing interest in scheduling for heterogeneous distributed systems is due to the dimensions of some large scale applications, which makes inadequate a single parallel architecture to cover their needs for parallelism.. When dealing with a computational Grid for parallel and distributed computing we have to efficiently exploit the computational power. In many practical cases, heterogeneous systems have proved to produce higher performance at lower cost than a single high performance computing machine. Scheduling applications on wide-area distributed systems is useful for obtaining quick and reliable results in an efficient manner. Optimized scheduling algorithms for multi-criteria constraints are fundamentally important in order to achieve advanced resources utilization. The modern applications address many fields of activity like satellite data processing, medicine, and others.

For example, MedioGRID is a successfully research project aiming at implementing a real-time satellite image processing system for extracting relevant environmental and meteorological parameters using a Grid system [1]. An essential requirement for a global Data Grid is to support the parallel and distributed processing of huge data. Therefore, it requires a scheduling system. The system needs to access and process the satellite image archives; the job manager should assign the jobs to available resources, basically by splitting the image in sub-images and process each sub-image on a different node.

Other applications relying on advanced scheduling solutions and involving workflows address the field of medicine. For example the EMAN application (Electron Micrograph Analysis), a bio-imaging workflow application deals with 3D reconstruction of single particles from electron micrographs [2]. The refinement from a preliminary 3D model to the final 3D model is fully automated and is the most computationally intensive step that benefits from exploiting the power of the Grid.

Such practical requirements ask for the development of optimized scheduling procedures to better benefit from the computational power of Grids. In this paper we present a novel performance analysis solution for DAG scheduling algorithms, based on modeling and simulation. First, we analyze the task dependencies (DAG) model and present a series of

modern algorithms that have near optimal performance capabilities. We present next a series of DAG scheduling algorithms that address the scheduling for a wide-range of task models, implying various time and resource requirements, on different distributed systems architectures. We continue with the description of several Grid simulation tools, highlighting the reasons for selecting MONARC as the simulator to use for analyzing the performances of DAG scheduling algorithms. Section 4 presents MONARC architecture and its functionality. In section 5 we present an original approach to evaluating DAG scheduling algorithms using the MONARC simulator. Performance analysis and expected experiments of this approach are presented in section 6. Finally, in Section 7 we present some conclusions and future work.

2. DAG Scheduling Algorithms

DAG scheduling is a NP-complete problem [3]. A solution to solving this problem is to use heuristics and task assigned priorities. The method is to select the task with the higher priority and give it access to resources before those tasks with lower priorities. The heuristics used vary according to job requirements, structure and complexity of the DAG [4], [5], [6]. Two attributes are used for assigning priorities: *tlevel* (top-level) for a node *u* is the weight of the longest path from the source node (the first scheduled task) to *u*, and *blevel* (bottom-level) for a node *u* is the weight of the longest path from *u* to an exit node (the latest scheduled task).

Once the priority mechanism is established the scheduling algorithm must take into consideration those tasks having all dependencies solved (the tasks they depend on have been executed) and minimize the time associated to the critical path. For example, HEFT (Heterogeneous Earliest Finish Time) selects at each step the task with the highest "upward rank" (the maximum distance from the current node to the finish node given by the computational and communication cost).

Based on the selected algorithms, the main research issue is to analyze the performance of the scheduling criteria and if possible, optimize the task scheduling by combining the best characteristics of a set of algorithms.

The *HLFET (Highest Level First with Estimated Times)* algorithm is one of the simplest scheduling algorithms (proposed for parallel systems in [7]). The algorithm schedules a task to a processor that allows the earliest start time. The main problem with HLFET is that when calculating the Static Level (SL) of a node, it ignores the communication costs of the edges.

The *ETF (Earliest Time First)* algorithm [8] computes, at each step, the earliest start times of all ready nodes and selects the one having the smallest start time. Here, the earliest start time of a node is computed by examining the start time on all processors exhaustively. When two nodes have the same value of their earliest start times, the algorithm breaks the tie by scheduling the one with the higher *SL*. Thus, a node with a higher *SL* does not necessarily get scheduled first because the algorithm gives a higher priority to a node with the earliest start time.

CCF (Cluster ready Children First) is a dynamic algorithm. In this algorithm the graph is visited in topological order, and tasks are submitted as soon as scheduling decisions are taken [9]. The algorithm considers that when a task is submitted for execution it is inserted into the RUNNING-QUEUE. If a task is extracted from the RUNNING-QUEUE, all its successors are inserted into the CHILDREN-QUEUE. These queues can be prioritized where the priority is based on the *tlevel* and *blevel* parameters. For example the priority of a task depends on the *tlevel*(*u*) + *blevel*(*u*) factor.

The *Hybrid Remapper* ([10]) is a dynamic list scheduling algorithm specifically designed for heterogeneous environments. Like the other algorithms its starting point is an initial DAG labeled with the execution and data transfer times (obtained by an initial static matching and scheduling). Then, the algorithm executes in two phases. In the first phase the set of tasks is partitioned into blocks so that the tasks in a block do not have any data dependencies among them. However, the order among the blocks is determined by the data dependencies that are present among the tasks of the entire DAG (it is a kind of level decomposition). The second phase is executed during application run time and involves remapping the tasks considering the changes to the initial information (statically determined). The *Hybrid Remapper* algorithm considers the static priority.

The MCP (Modified Critical Path) algorithm [13] uses the *ALAP* (As Late as Possible) time of a node as a scheduling priority. The *ALAP* time of a node is determined by first computing the length of CP and then subtracting the *blevel* of the node from it. Thus, the *ALAP* times of the nodes on the CP are just their t-levels. The MCP algorithm first computes the *ALAP* times of all the nodes and then constructs a list of nodes in ascending order of *ALAP* times. Ties are broken by considering the *ALAP* times of the children of a node. The algorithm then schedules the nodes on the list one by one such that a node is scheduled to a

processor that allows the earliest start time using the insertion approach.

3. Grid Simulation Tools

The use of discrete-event simulators in the design and development of large scale distributed systems is appealing due to their efficiency and scalability. Their core abstractions of process and event map neatly to the components and interactions of modern-day distributed systems and allow designing realistic simulation scenarios. However, because of the complexity of the Grid systems, involving many resources and many jobs being concurrently executed in heterogeneous mediums, there are not many simulation tools to address the general problem of Grid computing.

SimGrid is a simulation toolkit that provides core functionalities for the evaluation of scheduling algorithms in distributed applications in a heterogeneous, computational Grid environment [15]. It aims at providing the right model and level of abstraction for studying Grid-based scheduling algorithms and generates correct and accurate simulation results. *GridSim* is a grid simulation toolkit developed to investigate effective resource allocation techniques based on computational economy. *OptorSim* [16] is a Data Grid simulator project designed specifically for testing various optimization technologies to access data in Grid environments. OptorSim adopts a Grid structure based on a simplification of the architecture proposed by the EU DataGrid project. *ChicagoSim* is a simulator designed to investigate scheduling strategies in conjunction with data location [17]. It is designed to investigate scheduling strategies in conjunction with data location.

These simulation instruments are either too focused on particular aspects of grids and distributed systems (such as data replication, scheduling), or not extensive enough, allowing the modeling of a limited number of possible scenarios. They all tend to narrow the range of simulation scenarios to specific subjects, without considering all the characteristics. There is little room for modeling experiments designed to test, for example, a newly proposed job scheduling procedure for tasks involving processing activities, as well as communications among them, as encountered in real-world situations.

For these reasons we considered the use of the MONARC simulator [11] to analyze the performance of Grid DAG scheduling algorithms such as the ones presented above.

MONARC is a generic simulation framework designed for modeling large scale distributed systems. It allows the realistic simulation of a wide-range of distributed system technologies with respect to their specific components and characteristics. As presented in [12], its simulation model includes the necessary components to describe various actual distributed system technologies, and provides the mechanisms to describe concurrent network traffic, evaluate different strategies in data replication, and analyze job scheduling procedures. MONARC can be used to simulate a large number of possible scenarios, ranging from high-granularity network protocols to large-scale distributed systems comprising of many type of resources, to the modeling and simulation of the behavior of various applications making it an ideal instrument to accurately evaluating the performances of DAG scheduling algorithms as the ones presented in this paper.

4. MONARC Architecture

MONARC is built based on a process oriented approach for discrete event simulation, which is well suited to describe concurrent running programs, network traffic as well as all the stochastic arrival patterns, specific for such type of simulation. Threaded objects or "Active Objects" (having an execution thread, program counter, stack...) allow a natural way to map the specific behavior of distributed data processing into the simulation program.

In order to provide an easy to use simulator, all the components of the system and their interactions were abstracted. The chosen model is equivalent to the simulated system in all the important aspects. A first set of components was created for describing the physical resources of the distributed system under simulation. The largest one is the regional center (see Figure 1), which contains a farm of processing nodes (CPU units), database servers and mass storage units, as well as one or more local and wide area networks. Another set of components model the behavior of the applications and their interaction with users. Such components are the “Users” or “Activity” objects which are used to generate data processing jobs based on different scenarios. The job is another basic component, simulated with the aid of an active object, and scheduled for execution on a CPU unit by a “Job Scheduler” object. Any regional center can dynamically instantiate a set of users or activity objects, which are used to generate data processing jobs based on different simulation scenarios. Inside a regional center different job scheduling policies may

be used to distribute jobs to corresponding processing nodes.

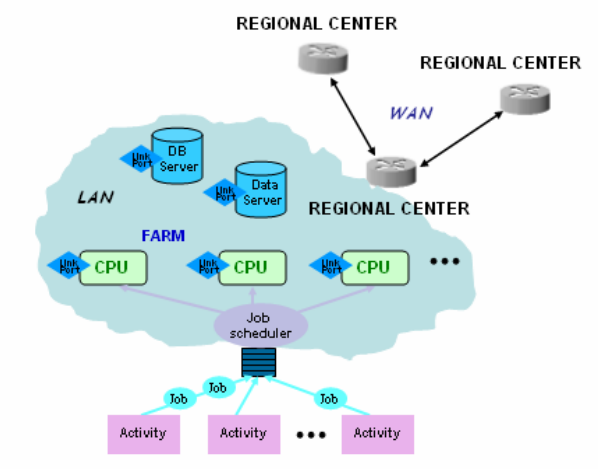


Figure 1. The Regional center model.

With this structure in mind it is possible to build a wide range of models, from the very centralized to the distributed system models, with an almost arbitrary level of complexity (multiple regional centers, each with different hardware configuration and possibly different sets of replicated data).

For resource scheduling, the grid simulator must be able to model the scheduling policies used by resource brokers to determine which resource should process an arriving task. The MONARC simulation model provides a meta-scheduling functionality, allowing the execution the simulated jobs in a distributed manner. The meta-scheduler can incorporate a wide-range of user-defined scheduling algorithms. Local scheduling algorithms can also be added in any user-defined experiments.

5. Simulating DAG Scheduling Algorithms using MONARC

With the MONARC simulation model, users can define various types of jobs to model common types of actions that can occur in any distributed systems: processing, data transfer (both pull and push models are supported), database handling, etc. The model is not limited to specific activities, the user having the possibility to easily incorporate new advanced job behaviors, as specified in the simulation scenario being executed. A simulated regional center also uses the services of a job scheduler. In order to schedule a job for execution the scheduler executes an appropriate scheduling algorithm.

The modelled job object contains a number of parameters that are used to estimate the time needed for execution. The time needed by a job to complete a CPU-intensive operation is estimated based on a number of attributes such as the CPU power, memory and the processing time needed to complete. For the data processing jobs, these attributes depend on the type of data that the job works with (in the configuration file, the user can set this parameters for each data type used in the simulation). Once the CPU-intensive job starts processing the time needed to complete its operation is pre-computed. If another job starts executing on the same processing unit before the first one completes, then an interrupt mechanism is used to handle the re-estimation of the time needed for both jobs.

The time needed for an I/O intensive job (for example, a data transfer handling type of job) is based on the mechanism provided by the network model. In this case again an interrupt mechanism is used to simulate the competition for bandwidth usage of concurrent data transfer jobs.

Within the job model the user can define new jobs starting from the basic behaviour provided. It can even combine several behaviours in one single composite job type. This aspect can be used to simulate a job that transfer some data, then processes it and in the end transfers back the obtained results. This behaviour represents a composition between the processing and data transfer types of jobs and can be modelled using only five lines of codes.

But, in the same type, the user can include a job which does all the data processing according to some advanced algorithm, extending in this case one method provided by the processing data type of job. Again, this can be accomplished with limited effort from the end-user.

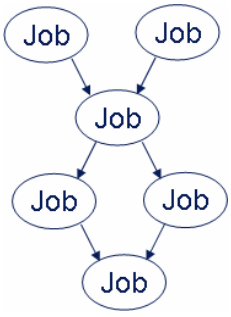


Figure 2. Job dependencies in MONARC.

One interesting aspect is the job decompositions being offered by the job model. The user can specify a situation where a job requests some data, and then split itself in several parallel tasks, each one processing a

particular chunk of data, and in the end the obtained results are reassembled and sent back. This fork-join programming paradigm can be modelled with the dependence mechanisms being offered.

Any job can instantiate new jobs. This means that, for example, one job receives the data, splits it into chunks and instantiate processing jobs, each one supplied with one specific chunk of data. It then specify the dependence, meaning it specifies what jobs must be executed after it finishes its own execution. The dependency between jobs can be specified in the form of a DAG structure (see Figure 2).

The simulation model also allows the evaluation of advanced scheduling algorithms such as the ones we were particularly interested in, evaluating DAG Grid strategies. However, in order to accommodate the modeling of the DAG scheduling algorithms we had to extend this default behavior of the job model in MONARC (see Figure 2).

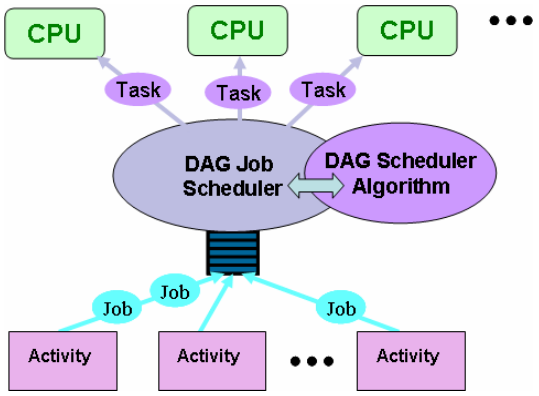


Figure 2. The extended scheduler modeling component.

First, the default scheduler behavior does not consider the topology of the underlying resources for scheduling assignments or the costs of the edges as in case of the algorithms presented in Section 2. Instead, the scheduler uses a highly simplistic algorithm that first prioritizes the tasks such that to eliminate all possible loops and then schedules each task to be executed according to its own locally requirements, without considering the minimization of the execution times among them. We extended the default behavior to allow for more realistic scheduling decisions that consider both the underlying topology (input parameter for the DAG job scheduler algorithm), as well as the full topology of tasks dependencies.

Another decision was related to the way user can specify the job scheduling algorithm. In order to test a new algorithm the user should present it to the job scheduler. We implemented this feature so that the user only needs to supply a corresponding DAG scheduling algorithm to be used by specifying the name of a class implementing a particular interface.

We also added an extension to the default job object. This was needed to allow a job to carry enough information about the corresponding input DAG topology as required by a DAG scheduling algorithm. We also added a new type of job in the simulation model to specify the particular needs to simulate a task with dependencies. This new job executes three actions: it first waits for data from all inner tasks, it then processes data according to the specified weights in the DAG, and it finally sends data to all corresponding outer tasks. In this way we take into account the complete values specified in a DAG scheduling situations, the costs of edges as well as the costs of all vertices.

Due to simulator's openness and modularity we were able to develop simulation scenarios using custom user-provided DAG scheduling algorithms to assigns tasks for execution on existing simulated resources. We added an extension to the default job component, allowing the specification of dependencies among tasks. The simulated tasks are modeling processing activities and the dependencies among them are modeled as data transfers among tasks. The scheduler receives as input sets of tasks with dependencies and, using the provided algorithm and the underlying modeled configuration of resources, it assigns each task to a corresponding processor for execution.

6. Performance analysis and expected experiments

Using MONARC's extensions we proceeded to evaluate the DAG scheduling algorithms presented in section 5. We evaluated several DAG scheduling procedures presented in section 2. We were particularly interested in analyzing their performance considering the use of two realistic scenarios.

For this reason the modeling experiment considered the use of two connected processors and a set of tasks with dependencies. The example is presented in Figure 3. For the experiments we considered four DAG scheduling procedures: CCF, ETF, HLFET and HybridRemapperPS (HybridPS). With the assumed conditions, the four experiments generated the theoretical scheduling decision of task assignments that are presented in Figure 4.

There are a number of factors that determine which algorithm is more suitable for a particular application. For example the schedule length when the application needs to execute as fast as possible, and the load balancing schedule with idle times reduced as much as possible. Taking too much time for the scheduling process is not always recommended especially in time critical applications. However, if there is a chance to improve the resulted schedule then there should be a compromise in order to run a more complex and time consuming algorithm to obtain better results.

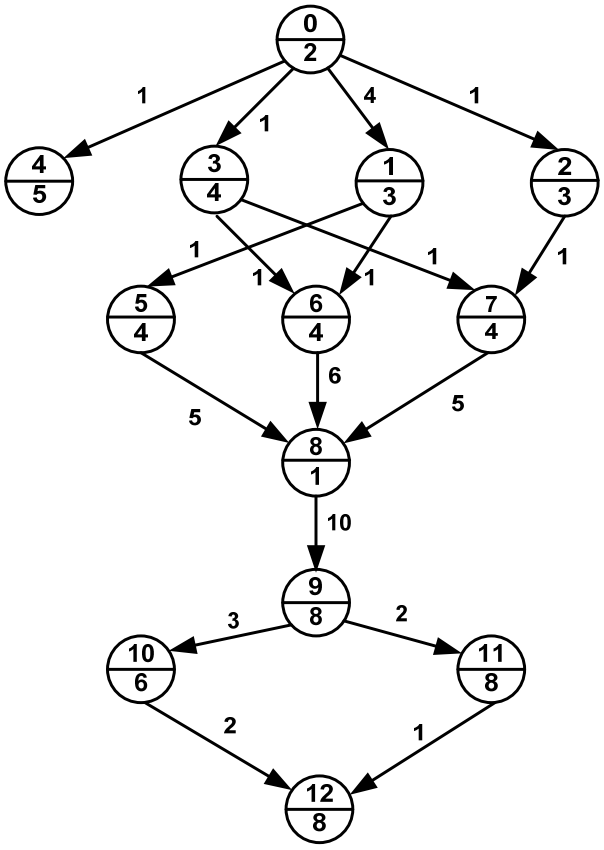


Figure 3. The DAG tasks used in the simulation experiments.

As it can be seen in the charts, the CCF algorithm offers the best load balancing and the minimum total time. ETF has the same total schedule time as HLFET.

However, the P2 processor is not used at all in this case (ETF), which means that for the same time we can reduce the number of processors just by using a different scheduling strategy.

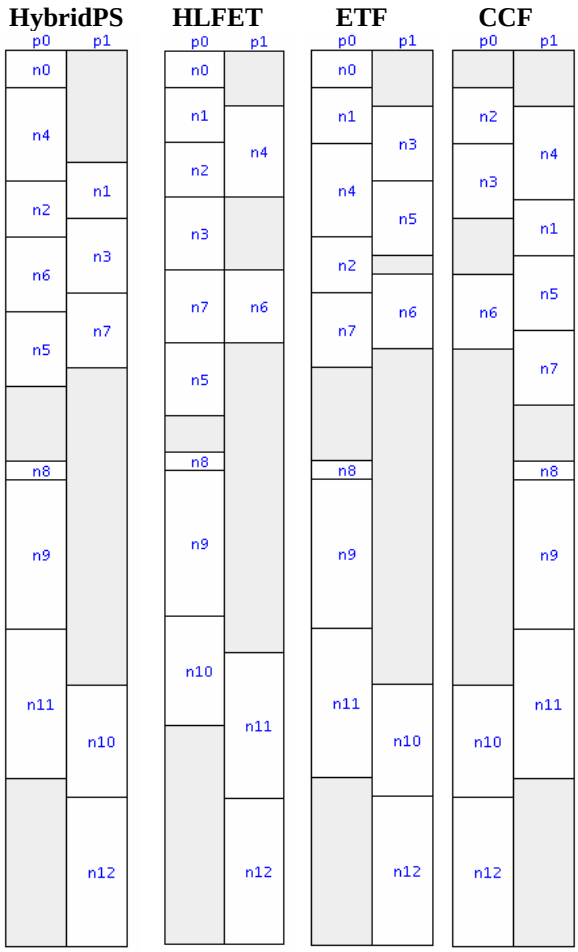
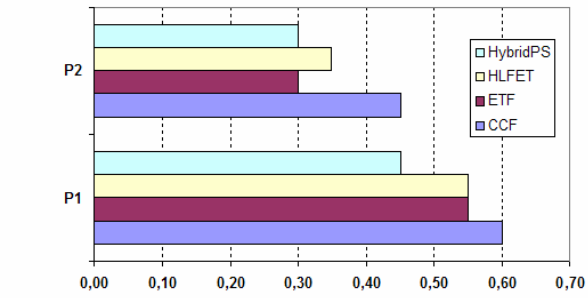


Figure 4. The scheduling decisions.

At the same time, for a given set of processors it might be inefficient to not use all the available resources (processors). Some of the tasks can be moved to a free processor thereby reducing scheduling time considerably.

We must also take in consideration that the above results have been obtained for a small number of tasks and a reduced number of processing resources. Further analysis on a larger and more complex DAGs could lead to different results for the total processing time and load balancing. We plan to further evaluate these aspects, including many more scheduling algorithms in the evaluation.

The CPU utilization in each of the four experiments is presented in Figure 5. The lowest CPU utilization was obtained for the CCF scheduling algorithm, which is explained by the higher number of transfers between different processors involved in the simulation experiment.



	P1	P2
HybridPS	0,45	0,30
HLFET	0,55	0,35
ETF	0,55	0,30
CCF	0,60	0,45

Figure 5. The CPU Utilization results obtained in each experiment.

The numbers of jobs concurrently running at any time in the modeled system are presented in Figure 6. The CCF algorithm has the best time. If the HLFET algorithm has an appropriate execution time for scheduled tasks, the CCF offers a better load balancing.

The other aspect presented in the Figure 6 is the *Tasks*Time* parameter (the area under the graphics). The CCF has the smallest value for this parameter. The *Tasks*Time* represents an average time of the CPU usage for the scheduled tasks. The CCF algorithm has the best resource utilization score.

The smallest period with maximum running tasks number is for the CCF algorithm. This shows a good load-balancing characteristic offered by this algorithm.

7. Conclusions and Future Work

Tasks with DAG dependencies are frequent in case of Grid applications and they require advanced scheduling procedures that must consider QoS requirements. In this paper we proposed a simulation-based solution to evaluate the performances of Grid scheduling algorithms.

The results could be used in decisions regarding optimizations to existing Grid DAG Scheduling and for selecting the proper algorithm for DAG scheduling in various actual situations.

The main contribution of the presented research consists in the development of the simulation layer in MONARC that is appropriate for DAG scheduling algorithms evaluation. We introduced a set of recent algorithms and presented the solution to evaluate DAG scheduling algorithms using a generic simulator for large scale distributed systems.

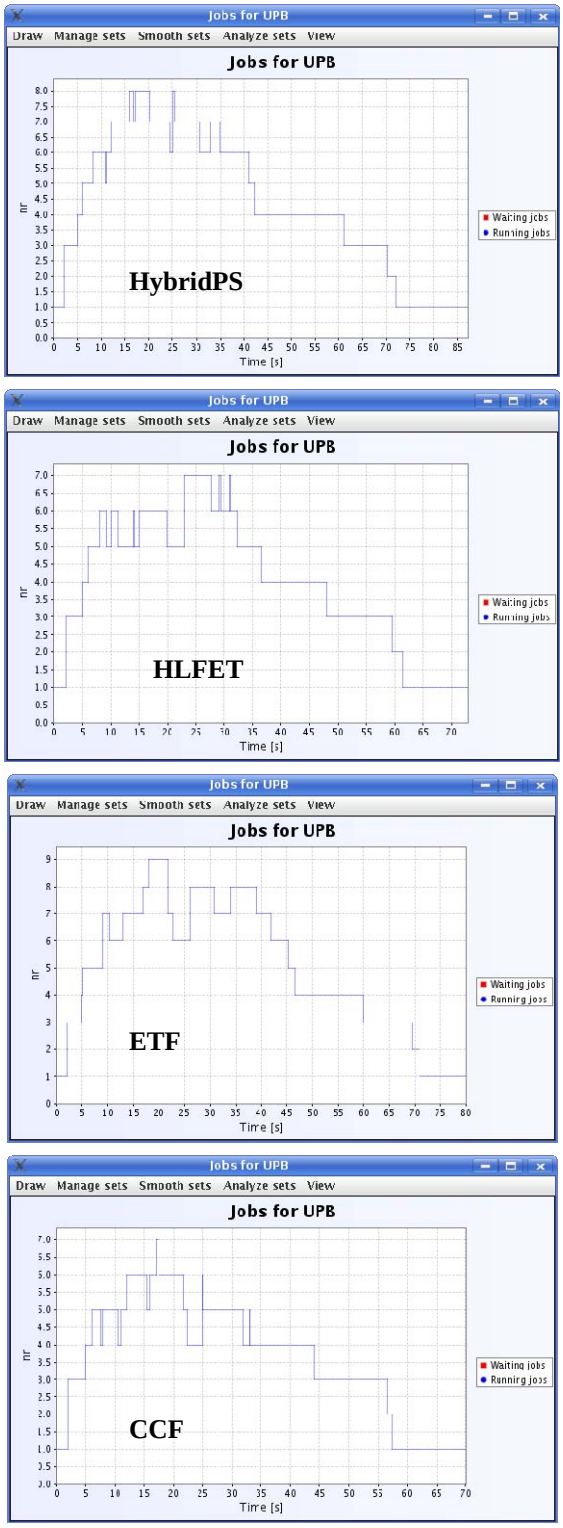


Figure 6. The obtained results in case of the number of concurrent jobs in the system for each modeling experiment.

The simulator represented an adequate instrument to approach a key problem of ensuring the high performance behavior of the Grid: the scheduling dependent activities.

Future work will include, among other things: the analysis of a wider set of scheduling algorithms currently used in Grid systems; the establishment of relevant performance measures and an improved simulation model; the evaluation of the current scheduling algorithms, using the chosen model.

We will consider new scheduling algorithms for real-time scenarios, solutions for backup and recovery from error (re-scheduling) and solving the problem of co-scheduling and multi-criteria constraints scheduling.

We also plan to use this performance analysis of DAG scheduling for the DIOGENES (DIstributed near-Optimal GENEtic Algorithm for grid application Scheduling) project extensions [14].

8. References

[1] Ovidiu Muresan, Florin Pop, Dorian Gorgan, Valentin Cristea, "Satellite Image Processing Applications in MedioGRID," *ISPDC*, pp. 253-262, Proceedings of The Fifth International Symposium on Parallel and Distributed Computing (ISPDC'06), 2006

[2] A. Mandal, A. Dasgupta, K. Kennedy, M. Mazina, C. Koelbel, G. Marin, K. Cooper, J. Mellor-Crummey, B. Liu, L. Johnsson, "Scheduling workflow applications in GrADS," *ccgrid*, pp. 790-797, Fourth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'04), 2004

[3] M. Kafil and I. Ahmad, *Optimal task assignment in heterogeneous distributed computing systems*, IEEE Concurrency 6, 3 (July-Sept. 1998), 42-51.

[4] D. Fernandez Baca, *Allocating modules to processors in a distributed system*, IEEE Trans. Software Engrg. 15, 11 (Nov. 1989), 1427-1436.

[5] C.-C. Shen and W.-H. Tsai, *A graph matching approach to optimal task assignment in distributed computing system using a minimax criterion*, IEEE Trans. Comput. 34, 3 (Mar. 1985), 197-203.

[6] O. H. Ibarra and C. E. Kim, *Heuristic algorithms for scheduling independent tasks on nonidentical processors*, J. Assoc. Comput. Mach. 24, 2 (Apr. 1977), 280-289.

[7] T. L. Adam, K. M. Chandy, and J. Dickson, *A comparison of list scheduling for parallel processing systems*, Comm. ACM 17, December 1974.

[8] J. J. Hwang, Y. C. Chow, F. D. Anger, and C. Y. Lee, *Scheduling precedence graphs in systems with inter-processor communication times*, SIAM J. Comput. 18, April 1999.

[9] FORTI A., DAG Scheduling for Grid Computing systems. Ph.D. Thesis, University of Udine – Italy, 2006

[10] M. Maheswaran and H. J. Siegel. A dynamic matching and scheduling algorithm for heterogeneous computing system. In the 7th Heterogeneous Computing Workshop (HCW '98), pages 57–69. IEEE Computer Society Press, March 1998.

[11] Dobre, C.M., C. Stratan, "MONARC Simulation Framework", *Proc. of the 3rd Edition of RoEduNet International Conference*, Timisoara, Romania, 2004.

[12] Dobre, C.M., V. Cristea, "A Simulation Model for Large Scale Distributed Systems", *Proc. of the 4th International Conference on Innovations in Information Technology*, Dubai, United Arab Emirates, November 2007.

[13] M.-Y. Wu and D. D. Gajski, *Hypercool: a programming aid for message-passing systems*, IEEE Trans. Parallel Distrib. Systems, July 1990.

[14] George Iordache, Simona Boboila, Florin Pop, C. Stratan, V. Cristea, *A decentralized strategy for genetic scheduling in heterogeneous environments*, Multiagent and Grid Systems, Volume 3, Number 4, ISSN: 1574-1702; DIOGENES project web page, http://diogenes.grid.pub.ro/ Accessed on 15 March 2008.

[15] Henri Casanova, Arnaud Legrand and Martin Quinson, *SimGrid: a Generic Framework for Large-Scale Distributed Experimentations*, Proceedings of the 10th IEEE International Conference on Computer Modelling and Simulation (UKSIM/EUROSIM'08).

[16] William H. Bell, David G. Cameron, Luigi Capozza, A. Paul Millar, Kurt Stockinger, and Floriano Zini. *OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies*. International Journal of High Performance Computing Applications, 17(4), 2003.

[17] F. Pop; C. Dobre; G. Godza; V. Cristea, *A Simulation Model for Grid Scheduling Analysis and Optimization*, Parallel Computing in Electrical Engineering, PARELEC 2006, Page(s):133 – 138.