Improving Workflows Execution on DAGMan by a Performance-driven Scheduling Tool

David Monge[1,3], Carlos García Garino[1,2]

[1] Instituto para las Tecnologías de la Información y las Comunicaciones (ITIC), UNCuyo
[2] Facultad de Ingeniería, UNCuyo
[3] PhD Fellowship CONICET
{dmonge, cgarcia}@itu.uncu.edu.ar

Abstract. In several scientific and business environments, the magnitude of problems require the execution of complex applications, like workflows. Condor includes DAGMan extension in order to manage workflows. Such kind of applications are very popular because jobs can be reused and distributed. Scheduling is a central issue in the assignation of resources to workflows. It can be seen as the process of assigning the resources to the workflow's jobs in a convenient way. This work presents a performance-driven approach to make these assignments (also known as mappings). The mappings generated determine in which resources the jobs must be executed. The execution stage is delegated to the DAGMan Workflow System. This approach is compared with the standalone execution of the workflow over DAGMan. Experimental results show that the performance-driven approach improves the execution time of workflows in comparison with the standalone DAGMan approach.

Keywords: Workflow, Scheduling, DAGMan, Performance, Optimization

1 Introduction

In the last years, the distribution of workflows over computational resources has become a central issue on large applications execution. Workflow applications are being widely used on scientific and business environments, where computing intensive programs are involved and a large amount of data must be processed. Due to different resources capabilities, different mappings of the workflow jobs onto the resources can be defined, each one with different execution costs. Search of the optimal mapping is an NP-Complete problem, then efforts are focused on the search of sub-optimal mappings as can be seen on a previous work of the authors [1]. These sub-optimal mappings are more costly than the optimal one but it can be obtained in much less time. In this work a scheduler for optimization of workflows makespan is presented. The scheduler is used to determine the mapping for the execution of the workflow applications. Once the mapping has been obtained, the workflow is expressed in terms of DAGMan (Directed Acyclic

Graph Manager) [2] for its execution on a Condor Pool [3,4]. The results of this performance-driven approach are compared with the results obtained by the standalone DAGMan execution.

The structure of this paper is as follows: a background of workflow scheduling and the DAGMan workflow manager is summarized on section 2, the explanation of our approach to solve scheduling and execution problems is discussed on section 3. Section 4 introduces the standalone DAGMan approach. On section 5 the definition of the experiments and the obtained results are explained, finally on section 6 the conclusions and future works are presented.

2 Background

2.1 Workflow Scheduling

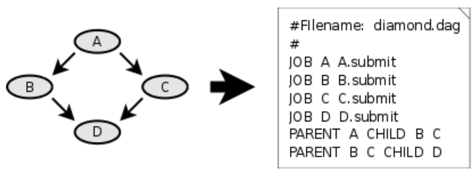
Workflow scheduling can be defined as the process of finding a mapping of workflows jobs onto available machines in a convenient way [5]. Scheduling of workflows is a complex task and different approaches were been developed. In the work [6] a theory for scheduling DAGs in Internet-Based Computing is introduced. In the work of Malewicz et al. [7], the authors discuss a tool for optimizing DAGMan workflows by prioritizing the jobs. Other approaches makes use of heuristic-based methods [8,9,10,1,11].

There are two kind of schedulers according to their Planning Scheme [5]. The first type of scheduler is the static scheduler, where the scheduling and execution stages are separated, and the scheduling stage precedes the execution stage. The second type is the dynamic scheduler, this type of schedulers combine both scheduling and execution stages in one. Dynamic schedulers are more flexible than static ones. Over dynamic schedulers it is easier to implement recovery mechanisms, rescheduling based on resource-state changes.

2.2 DAGMan Workflow System

DAGMan [2] is a Workflow Management System implemented as a dynamic scheduler for the execution of jobs based on their dependencies. DAGMan is part of the Condor project and extends the Condor Job Scheduler [3] to handle job dependencies. These sets of jobs and dependencies are known as workflows. DAGMan's workflows are *DAG-based*, meaning that the workflows can be represented by directed acyclic graphs (DAG). This workflow representation and others are discussed on [5].

In general, a DAGMan workflow is described on a file that contains the definition of the participant jobs and the dependencies among them. A small example can be seen on figure 1.



```
#Filename: diamond.dag
#
JOB A A.submit
JOB B B.submit
JOB C C.submit
JOB D D.submit
PARENT A CHILD B C
PARENT B C CHILD D
```

Fig. 1: Diamond DAG example.

There are two kind of jobs that can be used in DAGMan: *i)* Computation jobs; and *ii)* Data Placement (DaP) jobs. A Computation job is any batch application that can be processed on any of the universes offered by Condor and managed by the Condor's Scheduler. Stork Server allows the management of Data Placement tasks such as file transfers, storage space allocation and release. Note that DaP jobs were not used in this work.

Condor's Scheduler decides on which machine will be executed each one of the jobs as the result of a matchmaking process [12,13]. This process is done taking on account a resource *requirements* specification for the job and the characteristics of the available resources. The evaluation of a particular job's requirements expression determines what machines are capable to execute that job. The default requirements expression for a job constrains the Operating System, the Hardware Architecture, the Disk space and the Memory. Other requirement constraints can also be specified [14].

Among the candidate machines, the selection of the one that will handle a particular job can be decided based on a preference expression known as *rank*. The machine with the highest rank, is the machine selected for that job. It is worth mentioning that these decisions are done at a job level without considering the other jobs. This is known as *local decision making* [5] and it has the disadvantage that a good local decision can affect the performance of the entire workflow. So, improvements on DAGMan's workflow execution times can be made by having *global decision makings* in a previous scheduling stage.

3 Performance-Driven Workflow Scheduling

In order to reduce workflows execution times, is important to consider the execution(transfer) times of jobs(data). Because a lot of machines with different capabilities might be available to execute the jobs, the task of search for a mapping that minimizes the makespan of the workflow is an NP-Complete problem. In a previous work of the authors [1], the benefits of an approximation approach were discussed. The search of sub-optimal mappings allows to reduce drastically the execution times of the algorithm with minimal loses on the solutions' qualities (near-optimal approximate solutions). So, an approximation approach is a good candidate to be applied on the workflow scheduling problem.

The objective of this work is to present a scheduling tool for the reduction of workflows' execution times. This scheduler is part of a Performance-driven Scheduling Approach (PSA, for short).

The PSA process has three stages: *i)* Scheduling: determine workflow mappings with global considerations, made by the scheduler; *ii)* Translation: a simple process to express the workflow in terms of DAGMan; *iii)* Execution: the managing of the workflow, made by DAGMan. The process can be seen on figure 2.

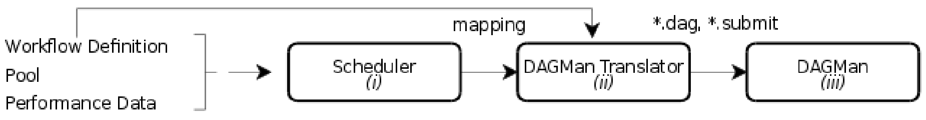


Fig. 2: PSA approach process.

3.1 Scheduling Stage

The tool presented in this work is a performance estimation based scheduler designed to search near-optimal mappings of jobs onto machines. The aim is to reduce the makespan of the workflows based on performance estimations. This accomplished by modeling the scheduling problem as a CSOP (Constraint Satisfaction and Optimization Problem). In order to reduce the complexity of the problem, the problem is simplified in a Pre-assignment stage. After that, the problem is solved, and the schedule (mapping) is obtained. A small schema of the Scheduler architecture is presented on figure 3.

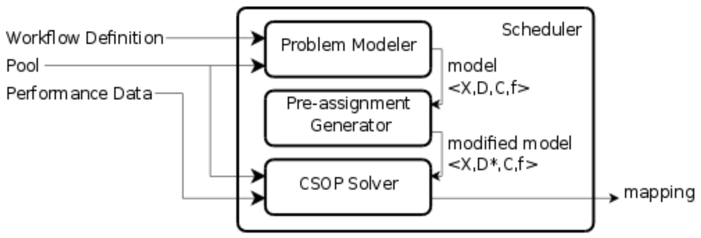


Fig. 3: Scheduler Architecture.

Modeling of the Scheduling Problem A scheduling problem is modeled as a 4-tuple $\langle X, D, C, f \rangle$, where:

- X is the set of variables: Variables to model workflow jobs (*job variables*) and new dependencies (*dependency variables*) involving two jobs.

- D is the set of domains: Each variable has its own domain. For *job variables*, domains are conformed by the possible machines on which that job can be executed. For *dependency variables*, domains are conformed by the values *after*, *before* and *independence*. The first two values establish the direction of the dependency, the third indicates that there is no relation between the jobs.
- C is the set of constraints: *Cycles constraints* avoids the generation of dependencies that produce cycles on the workflow DAG. *Cost constraint* is a soft constraint (it has a numeric value that points how good this variable assignment is in comparison with others) that calculates an estimation of the complete workflow execution time (makespan). *Overlap constraints* avoid the overlapping of jobs on a same machine at the same time.
- f is the optimization function: This is the function that will be optimized by the algorithm. In this work the optimization of the function is the makespan of the workflow $f = makespan$. So optimization of that function conduces us to the mapping that minimizes the total execution time of the workflow.

The solution for a scheduling problem is an assignment of the variables of the problem that satisfied all the constraints with the optimal value of f.

Pre-assignment Stage Once the problem is expressed in terms of the 4-tuple $\langle X, D, C, f\rangle$, this must be solved. In order to reduce the number of combinations of variable-values, a pre-assignment stage is carried out. This reduction of the domains conduce us to a reduction of the complexity of the problem. A new 4-tuple $\langle X, D^*, C, f\rangle$ is generated with the reduced domains.

The domains selected to be reduced are those corresponding to *critical jobs*. For this kind of jobs the domain is limited to the fastest machine. Critical jobs are jobs that if they are delayed the makespan of workflow is delayed too. Critical jobs have a *slack* equal to 0. This value is calculated as $slack = maxEndTime - startTime$. Where $maxEndTime$ is the maximum end time of the job that do not delay the makespan. And $startTime$ is the start time of the job. The slack represents how much time a job can be delayed without delaying the workflow makespan. All critical jobs conform a *critical path*.

The calculation of the jobs' *start* and *end* times are made as an estimation considering an resources average capacity. The details of how this performance information is used will be seen later on the subsection Performance Estimation.

Scheduling Algorithm As seen, the scheduler was implemented as a CSOP (Constraint Satisfaction and Optimization Problem) solver based on Backtracking and Branch & Bound algorithms mixture. This algorithm was improved with the use of heuristics for the pruning of the expansion tree generated by the Backtracking algorithm. The less nodes to expand, the faster execution of the algorithm to find the solution. The use of the algorithm without heuristics cannot be considered because workflow scheduling problems are NP-Complete problems.

Details about the scheduling tool can be found at previous works of the authors [1,11]. Some of the heuristics were introduced on the work [11]. On it, different combinations of heuristics were tested and three of them were selected as candidates for scheduling problem based on: the number of expanded nodes and execution time of the algorithms; and the proximity to the optimal solution. On this work a new heuristic for variable selection was developed (MRV for critical-earlier jobs).

The heuristics set used for the scheduler on this work are:

- *MRV heuristic prioritizing critical and earlier jobs (Variable Selection):* This heuristic selects the jobs to be mapped ordered by a 3 aspects measure: *i)* selects first, jobs that can be mapped on a minor number of machines (MRV). *ii)* selects first jobs that are critical (jobs with the minimum *slack*). *iii)* selects first jobs that are earlier on the workflow.
- *Max Capacity heuristic (Domain Ordering)*: Selection of the fastest machines first for a particular job [11].
- *Forward Checking (Constraints Propagation):* The removal of inconsistent values on variable domains on each assignment. For more details about the techniques and heuristics, you can see [11].
- *Sch-A (Performance Estimation)*: An approximation heuristic of the execution performance of the jobs and the transfers performance [11].

Performance Estimation In order to estimate the execution time of the different mappings a Performance Model is used. The calculation of the total execution time of the workflow is done with the use of the DAG representation of the workflow

The scheduler makes an intensive use of the DAG representation of the workflow. The DAG representation is used to handle the new dependencies and for make the performance estimations of the workflow. That representation is constructed with components of the JUNG framework [15].

The Performance Model is used to estimate the execution times of the jobs and the transfer times of the messages to be transferred. The estimation of execution times is made based on equation 1.

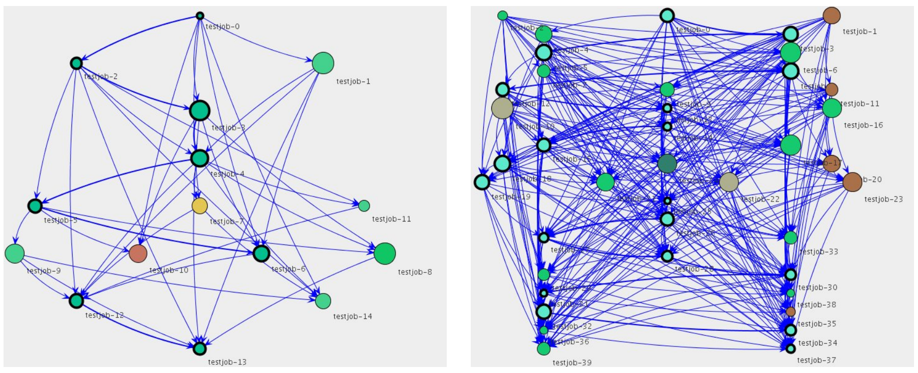
$$T_{ex}(job, machine) = PR_{job}/PC_{machine} \tag{1}$$

Where PR_{job} denotes the processing requirement of the job measured in a proper unit (MIPS for instance) and the processing capacity of the assigned resource to the job is denoted as $PC_{machine}$. Te estimation of transfer times is made based on equation 2.

$$T_{tx}(message, A, B) = size(message)/TR_{machineA,machineB} \tag{2}$$

Where $size(message)$ is the size of the message to be transmitted from machine A to machine B and $TR_{machineA,machineB}$ is the transfer rate between those machines.

Visualization In order to see the mapping obtained by the scheduler, the JUNG framework was used [15]. This is a Java framework for the analysis and visualization of several kinds of graphs. Figure 4a shows the mapping of a workflow with 15 jobs, and figure 4b shows the mapping a workflow with 40 jobs. Each circle represents a job, the colors indicate resources on which the machines are executed.

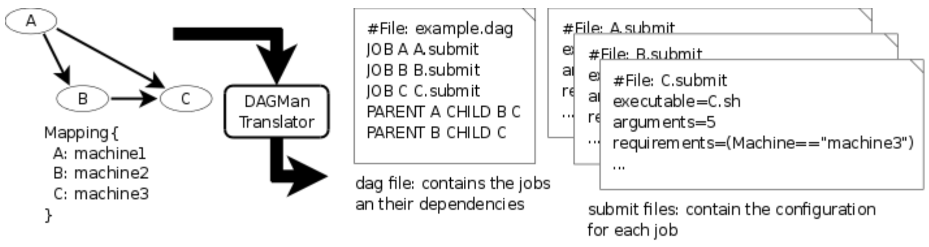


(a) Mapping of a workflow with 15 jobs. (b) Mapping of a workflow with 40 jobs.

Fig. 4: Workflow mappings visualization examples.

3.2 Translation and DAGMan's Execution Stage

Once the mapping has being determined, the execution of the workflow on DAGMan begins. The tool generates a DAGMan's submit file based on the *workflow definition* and the *mapping* obtained as result of the scheduling stage. For each job of the workflow, a Condor's submit file is generated. This file contains all execution configuration information for the job: executable file, arguments, input and output files, etc. Each job is also constrained to be handled by the resulting machine of the scheduling process. This is made by the definition of the expression $requirements = (Machine == the.mapped.machine)$. By doing this, the Condor's matchmaking process is reduced to the handling of the job on the specified machine. A simple example can be seen on figure 5.



```
Mapping{
  A: machine1
  B: machine2
  C: machine3
}
```

```
#File: example.dag
JOB A A.submit
JOB B B.submit
JOB C C.submit
PARENT A CHILD B C
PARENT B CHILD C
```

dag file: contains the jobs an their dependencies

```
#File: A.submit
```

```
#File: B.submit
```

```
#File: C.submit
executable=C.sh
arguments=5
requirements=(Machine=="machine3")
...
```

submit files: contain the configuration for each job

Fig. 5: DAGMan translator example.

Once the translation has finished, the dag is submitted to DAGMan for its execution. DAGMan submits all the ready-to-execute jobs to Condor according to the dependencies. Once all the parents of a job have finished, the job is submitted to Condor. This process continue until all jobs have finished.

4 The Standalone DAGMan Approach

In order to test the performance of the PSA approach, explained on section 3, the execution of workflows on DAGMan was carried out. The standalone DAGMan approach process has two stages: *i)* Translation, and *ii)* Execution. The process can be seen on figure 6.

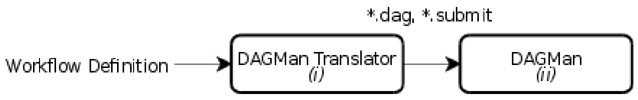
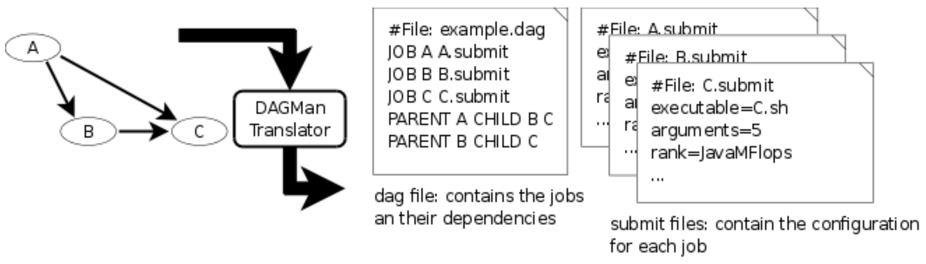


Fig. 6: Standalone DAGMan approach process.

The generation of the DAGMan's files that describe the workflow to be executed is made with the DAGMan Translator used in the PSA approach. But this time, the translation is made based only on the *workflow definition*. For the Condor's submit files of each job, the *requirements* expression is not set, allowing Condor to select the machine for each job. The *rank* expression is setted to $rank = JavaMFlops$, indicating to Condor the preference of machines with more processing capacity (more MFlops).


A
B
C
DAGMan Translator
dag file: contains the jobs an their dependencies
submit files: contain the configuration for each job

```
#File: example.dag
JOB A A.submit
JOB B B.submit
JOB C C.submit
PARENT A CHILD B C
PARENT B CHILD C
```

```
#File: A.submit
```

```
#File: B.submit
```

```
#File: C.submit
executable=C.sh
arguments=5
rank=JavaMFlops
...
```

Fig. 7: DAGMan translator example.

5 Experiments and Results

To study the benefits of the PSA execution approach over a pure DAGMan approach, a set of tests was designed. For both approaches, were executed workflows with a different number of jobs and dependencies between them. These workflows were generated randomly to consider several kinds of workflows in the experiments.

5.1 Random Workflow Generation

A workflow can be characterized on it's structure by two parameters. One of these is n, the number of jobs of the workflow. The other is δ, the dependencies density factor. δ represents the ratio of the number of dependencies of a workflow and the maximum possible ones. δ is a measure of the degree of independence between jobs. Lower values of δ means more independence between the jobs, and higher values of δ means more dependencies among them. Workflows with lower values of δ are often harder to map. The number of dependencies of a workflow can be determined by the δ factor. Let be mdn the maximum possible dependencies number on a workflow with n jobs, mdn can be calculated as:

$$mdn = n \times (n-1)/2 \tag{3}$$

Then, the dependencies number dn of a workflow with a particular value of δ can be calculated as:

$$dn = \delta \times mdn \tag{4}$$

For tests, workflows with $n = 5 \ldots 50$ step by 5 and $\delta = 0.4 \ldots 0.8$ step by 0.2 were randomly generated. For each workflow family defined by $\langle n, \delta \rangle$, 5 workflows were generated. The total number of workflows solved is $10 \times 3 \times 5 = 150$, these workflows were executed with the two approaches presented in this work.

Each workflow job is generated with an associated number of required operations to be completed, that number represents the amount of processing tasks needed by that job to finish. The number of operations for the jobs were defined in a range between 10000 and 60000. Each dependency is related to a message to be transferred between the dependent jobs. Sizes of each message goes between 10 and 30 MiB.

5.2 Workflow Jobs

Each one of the jobs used in the experiments is a Java program that simulates a real job that could be executed as part of a workflow. These jobs are called *Test Jobs.* The aim of a test job is to check the existence of the input files, to make some calculation operations and to write a set of output files to be transferred to other jobs.

Each job receives as arguments:

1. *number of operations*: is the number of operations to execute, each operation consist on 10000 random numbers generation, accumulation on a variable and division.

2. *list of input files*: the names of the files to be read by the program. Searched on the programs directory.
3. *list of local input files*: the names of the files to be read by the program. Searched on a machine's local directory. This is used when both parent and child jobs were mapped on the same machine. In this case there is no need to transfer the file, so it's kept on the machine. This issue is exploited only in the PSA approach because the source and target machines are known beforehand.
4. *list of output files*: the names and the sizes of the generated output files. Searched on the programs directory.
5. *list of local output files*: the names and the sizes of the generated output files. Searched on a machine's local directory as the same as the local input files.
6. *local directory*: the directory on the machine that executes the program, on which the local input files will be read and the local output files will be written.

5.3 Performance Data

In order to estimate the execution and transfer times of different jobs and messages respectively, performance data is needed as been explained on the Performance Estimation subsection. To use Performance Model, the processing capacity of machines and the transfer rates must be known. To retrieve this information, a benchmarking process must be carried out.

The processing capacity of a machine can be tested executing a test job with a big number of operations to execute. The processing capacity of the tested machine can be calculated as $PC = Ops/T_{test}$. Where Ops is the number of operations executed by the Test Job and T_{test} is the time used by the machine to complete the operations. A number of 10000 operations was used to estimate the processing capacity of the machines.

To estimate transfer times of messages an estimation of the transfer rates are needed. For this work the nominal values of the interconnection hardware were used. To a more complete and accurate test, a benchmarking tool should be used.

The characteristics of the machines pool are shown in table 1. Machine name, processor, memory and the processing capacity resulting of the benchmark process are listed:

Table 1: Processing Capacity Benchmark results.

Machine	Processor	Memory	PC $[ops/s]$
compute-0-2 (storm cluster)	Intel P4 HT 3.0GHz	1GiB	598.616
compute-0-3 (storm cluster)	Intel P4 HT 3.0GHz	1GiB	598.950
compute-0-4 (storm cluster)	Intel P4 HT 3.0GHz	1GiB	598.871
opteron0 (opteron cluster)	AMD Opteron 242 1.6GHz	2GiB	995.161
opteron1 (opteron cluster)	AMD Opteron 242 1.6GHz	2GiB	1079.471
compute-twister-1 (twister cluster)	Intel Core2 Duo 3.0Ghz	4GiB	1784.189

The interconnection hardware between all the machines is 100Mibps. This is the value used by the scheduler to make the transfer times estimation.

5.4 Results

In the figures the average total execution times of the both approaches are presented. The results are grouped by the dependency density factor (δ). There is one figure per each value of δ, and the average total times for each approach are presented ordered by the workflows' numbers of jobs (n). For the scheduled approach, the total time is calculated as the scheduling time plus the execution time of the workflow. For the pure DAGMan approach the total time is equal to the execution time of the workflow.

In the figure 8 the results for the $\delta = 0.4$ are presented. It can be seen that in all cases, the PSA approach has the lower total time. The figure also shows the standard deviation of the total execution times for both approaches. In general, the deviations are small, this means that the different workflow instances with the same number of jobs (n) have had the same behavior.

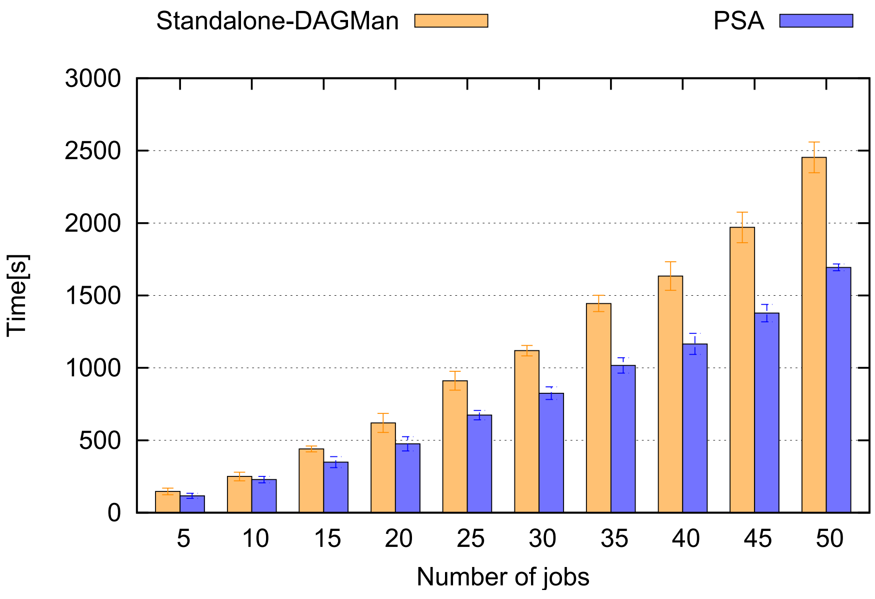


Fig. 8: Total executions times of workflows with $\delta = 0.4$.

In the figure 9 the results for the $\delta = 0.6$ are presented. Similar results were obtained for this tests set. In all cases, the PSA approach has the lower total time. The standard deviations are small in all cases.

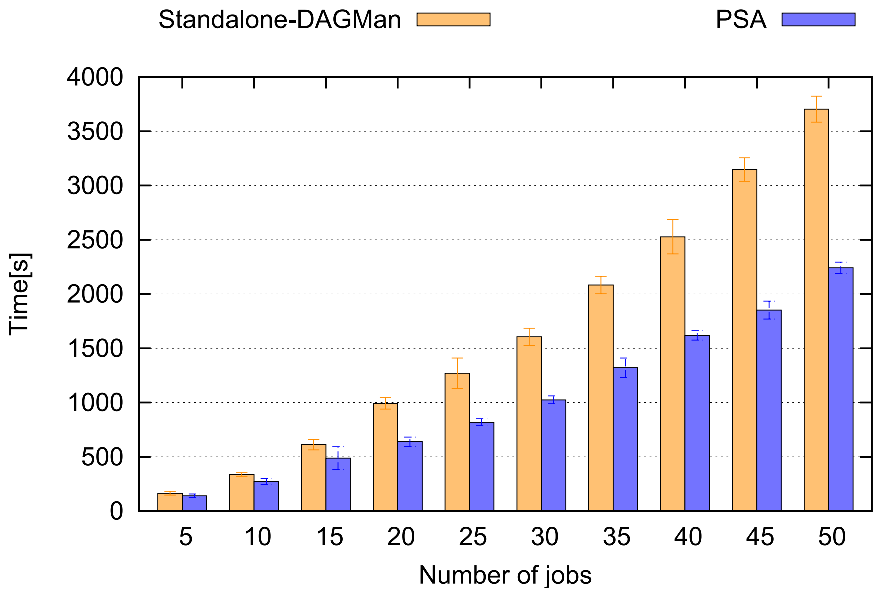


Fig. 9: Total executions times of workflows with $\delta = 0.6$.

In the figure 10 the results for the $\delta = 0.8$ are presented. The results of this tests set are similar to those obtained in the previous tests sets. In all cases, the PSA approach has the lower total time. The standard deviations are small in all cases.

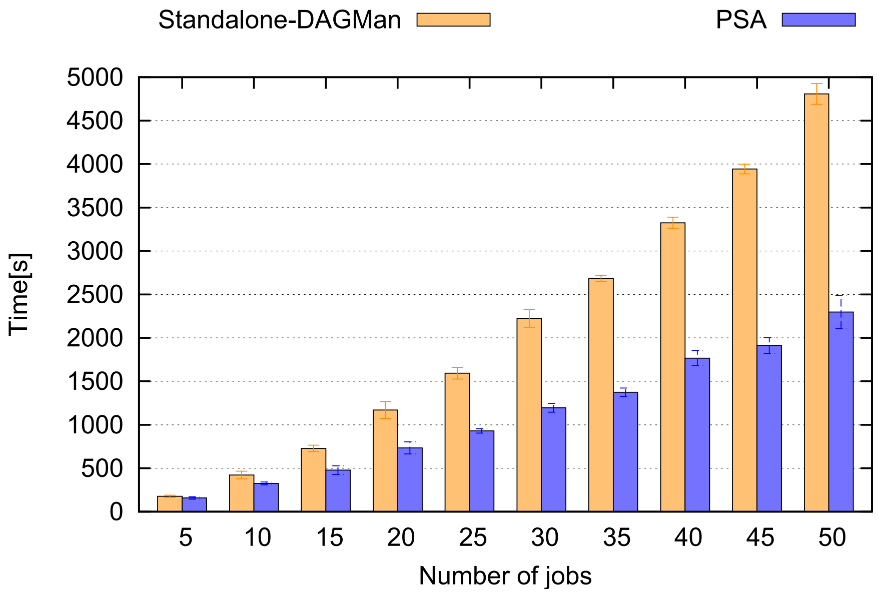


Fig. 10: Total executions times of workflows with $\delta = 0.8$.

In order to compare both approaches the speedup measure is used. The speedup for each group of workflows with the same values for δ and n, is calculated as:

$$S_{n,\delta} = \frac{T_{n,\delta}^{DAGMan}}{T_{n,\delta}^{PSA}} \tag{5}$$

Where $T_{n,\delta}^{DAGMan}$ is the average execution time of workflows with n number of jobs and a density of δ, using the Standalone DAGMan approach. And $T_{n,\delta}^{PSA}$ is the average execution time of workflows with n number of jobs and a density of δ, using the PSA approach. In figure 11 the speedups for workflow families with densities 0.4, 0.6 and 0.8 ($wf_{\delta=0.4}$, $wf_{\delta=0.6}$ and $wf_{\delta=0.8}$) are presented.

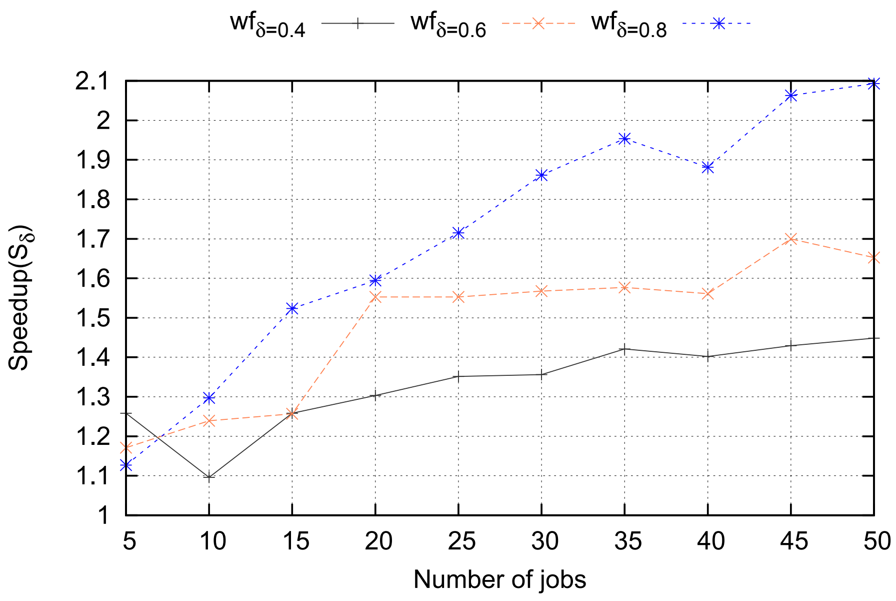


Fig. 11: Speedups obtained for each workflow family.

For the three families of workflows, the more jobs composing the workflow the better the speedup. This is because if the workflow has more jobs, the influence of the bad selection of resources by Condor becomes more evident.

In order to obtain a representative measure of all workflows with a particular value of δ. The average speedup for δ is calculated as:

$$S_\delta = avg(S_{n,\delta}) \quad n = 5 \ldots 50 \, step \, by \, 5 \tag{6}$$

The average speedups obtained for the three values of δ were: $S_{0.4} = 1.333$, $S_{0.6} = 1.483$ and $S_{0.8} = 1.711$. Note that in general the best speedups are obtained on the execution of more linear workflows (is to say for largest δ in this case). That is because more linear workflows have longer critical paths (in number of jobs), then, errors introduced by the selection of non propriate machines have a larger influence on the performance degradation. Speedups calculated to compare both approaches can be expressed as percentages of the improvement of workflows' execution times according to expression: $perc(S) = (1 - 1/S) \times 100\%$. In table 2 some important results are presented.

Table 2: Average peedups and average, max and min speedup percentages.

δ	S_δ	$perc(S_\delta)$	$min\{perc(S_{n,d})\}$	$max\{perc(S_{n,d})\}$
0.4	1.333	24%	9%	31%
0.6	1.483	31%	15%	41%
0.8	1.711	39%	11%	52%

6 Conclusions and Future Works

- The PSA approach improves the execution time of workflows in comparison with the execution of the same workflows over the standalone DAGMan. An average speedup of 1.333 was obtained for workflows with $\delta = 0.4$. For workflows with $\delta = 0.6$ an average speedup of 1.483 was obtained. And for workflows with $\delta = 0.8$ an average speedup of 1.711 was obtained. It can be seen that when δ increases (more linear workflows with more communication) the speedup obtained by the PSA approach increases.
- The smallest execution time gain is about an 9%, and the largest is about an 52%. The average execution time reductions obtained were 24%, 31% and 39% for workflows with densities $\delta = 0.4$, $\delta = 0.6$ and $\delta = 0.8$, respectively.
- The scheduling stage necessary for the PSA approach doesn't introduce a big overhead in the total execution time of the workflow. In the worst case, the overhead introduced is about 507ms, reducing the total execution time from 5023s (standalone DAGMan) to 1997s (PSA).
- The disadvantage of the PSA approach is that is difficult to implement rescheduling issues.
- A new performance estimation method must be considered in order to make estimations for other kinds of jobs.

7 Acknowledgments

The first author want to thank CONICET, for the PhD fellowship granted. The financial support provided by ANPCyT through PAE-PICT 2312 project and MINCyT Czech Republic bilateral cooperation RC0904 project is gratefully acknowledged.

References

1. David Monge and Carlos García Garino. A constraint optimization based scheduler for distributed computing workflows. In *Proceedings of the First Symposium on High-Performance Computing (HPC2009) in Latin America, 38 JAIIO*, Mar del Plata, Argentina, 2009.
2. Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the Grid. In *Grid Computing: Making the Global Infrastructure a Reality*, pages 11–33, 2003.
3. Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny. Condor: a distributed job scheduler. pages 307–350, 2002.
4. Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed Computing in Practice: The Condor Experience. *Concurrency and Computation: Practice and Experience*, 17:2–4, 2005.
5. Jia Yu and Rajkumar Buyya. A Taxonomy of Workflow Management Systems for Grid Computing, Apr 2005.
6. Grzegorz Malewicz, Arnold L. Rosenberg, and Matthew Yurkewych. Toward a theory for scheduling dags in internet-based computing. *IEEE Transactions on Computers*, 55:757–768, 2006.
7. G. Malewicz, I. Foster, A.L. Rosenberg, and M. Wilde. A tool for prioritizing dagman jobs and its evaluation. *High-Performance Distributed Computing, International Symposium on*, 0:156–168, 2006.
8. Suraj P, Linlin Wu, Siddeswara Mayura Guru, and Rajkumar Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments, 2010.
9. Yingchun Yuan, Xiaoping Li, Qian Wang, and Xia Zhu. Deadline division-based heuristic for cost optimization in workflow scheduling. *Inf. Sci.*, 179(15):2562–2575, 2009.
10. Maria M. Lopez, Elisa Heymann, and Miquel A. Senar. Analysis of dynamic heuristics for workflow scheduling on grid systems. In *ISPDC '06: Proceedings of the Proceedings of The Fifth International Symposium on Parallel and Distributed Computing*, pages 199–207, Washington, DC, USA, 2006. IEEE Computer Society.
11. David Monge and Carlos García Garino. Heurísticas novedosas de constraint optimization aplicadas al scheduling de workflows de computación distribuida. In *Proceedings of V Encuentro de Investigadores y Docentes de Ingeniería, EnIDI 2009*, Los Reyunos, San Rafael, Argentina, 2009.
12. Rajesh Raman. *Matchmaking frameworks for distributed resource management*. PhD thesis, 2000. Supervisor-Livny, Miron.
13. Rajesh Raman, Miron Livny, and Marvin H. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *HPDC*, pages 140–, 1998.
14. *Condor manual: http://www.cs.wisc.edu/condor/manual/v7.4/index.html*.
15. J. Madadhain, D. Fisher, P. Smyth, S. White, and Y.B. Boey. Analysis and Visualization of Network data Using JUNG. *Journal of Statistical Software*, 10:1–35, 2005.