A Constraint Optimization based Scheduler for Distributed Computing Workflows

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] Fellow CONICET
{dmonge, cgarcia}@itu.uncu.edu.ar

Abstract In this work, the scheduling of distributed computing workflows is discussed. In practice there are problems like Datamining applications were rather complex workflows are found. The mapping of workflows on the available resources has to satisfy matchmaking of job requirements and resources capacities. On the other hand job dependencies exists due to workflow characteristics. Then different constraints must be accomplished and Constraint Satisfaction Problems (CSP) is a good candidate in order to obtain the feasible set of solutions. In this case, Backtracking and Branch and Bound algorithms are considered. After the CSP problem is defined, the best solution or a close enough approximate one, have to be obtained. In this work the cost function is written in terms of workflow total execution time. From the validation experiments tried can be said that good quality solutions can be found. Moreover, approximate case, but saving important computational effort in order to find the solution.

1 Introduction

Distributed Computing techniques have been used very often in the last years in order to solve different applications. In some cases High Performance Computing [1,2] require large amount of computing facilities. Other applications are based in a rather complex workflow [3] of applications like Datamining for instance [4]. Finally Grid Computing [5,6] promises to provide required resources in a transparent way to users.

In order to deploy a distributed computing infrastructure different issues must to be addressed in practice, for instance resources have to be discovered and the scheduling of job resources have to be done.

In this work different techniques are discussed in order to optimize the workflow mapping to available resources.

Workflow scheduling is a rather complex problem. First a matchmaking between jobs and resources have to be executed. After matchmaking different possible assignments can be made in practice.

There are different restrictions to be accomplished like job dependencies, processing requirements, software disponibility, among other issues. Consequently to find a proper assignment of jobs to resources belong the Constraint Optimization Techniques.

In this paper Constraint Satisfaction Problem is used to approach workflow scheduling. In this way a modified backtracking algorithm is proposed.

In section 2 some background is provided. Workflow, Workflow Scheduling and CSP are addressed in subsections 2.1, 2.2 and 2.3 respectively. In section 3 Distributed Computing Workflow Optimization is studied and the algorithm is discussed.

Validation experiments are performed and its results are accounted in section 4. Finally Conclusions are provided in section 5.

2 Background

2.1 Workflow

Workflow can be defined as the flow of jobs and data required in order to execute an application. A workflow can be considered as an abstraction of an actual job with a well defined set of tasks and their dependencies. A workflow can be represented as a *DAG (Directed Acyclic Graph)* or a *non-DAG*. In figure 1 can be seen a DAG Workflow example.

According to Yu and Buyya [3] a DAG-based workflow structure can be classified as model sequence, parallelism and choice. First case is typical of serial applications, parallelism accounts for tasks that can be processed concurrently and choice let some options to be selected at run-time. A non-DAG workflow additionally includes iteration structure. In this sense one or more tasks are allowed to be repeated. Scientific applications often uses this kind of structures also known as loop or cycles.

An application workflow is a particular type of workflow where all the jobs are applications. For this kind of workflows tasks dependencies come from I/O data jobs relationships.

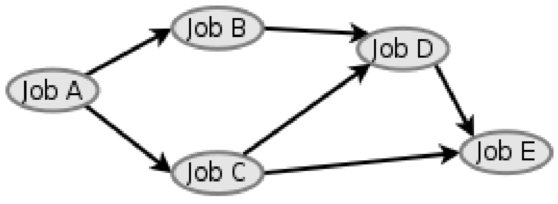


Figure 1. DAG workflow example.

2.2 Workflow Scheduling

Workflow scheduling can be considered as a kind of global job scheduling because it is focused on mapping and managing the execution of interdependent jobs on shared resources that are not directly under its control [3].

In practice the applications that compose the workflow have different requisites respect to the available resources. In order to state jobs requirements are usually taken into account: operating system, RAM memory, processor type, hard disk capacity, libraries and software availability, etc.

The goal of Workflow scheduling is to assign resources to applications and to define the proper execution instant as well. In figure 2 a simple example can be observed.

In general a single best solution for mapping workflows onto resources for different applications is rather difficult to be found. Then a decision making process must to be followed in order to schedule the workflow. An overview of the problem is described by Yu and Buyya [3] and references therein.

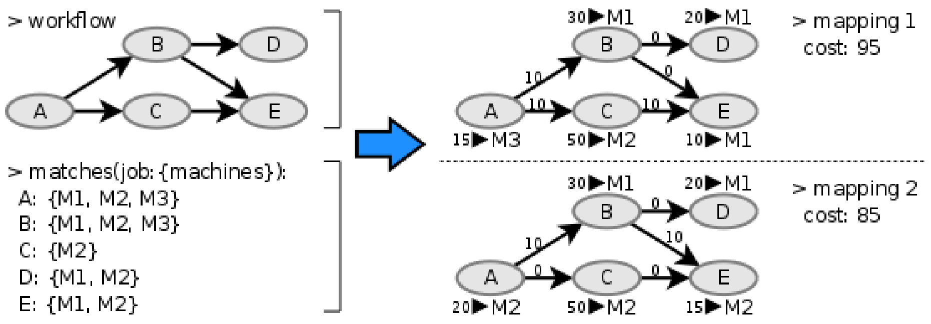


Figure 2. Scheduling example.

2.3 Constraint Satisfaction Problems

In Constraint Satisfaction Problems (CSP) [7] a set of values have to be assigned to the problem's variables in order to satisfy the constrain set. Formally a CSP can be defined in terms of the $P(X, D, C)$ where:

- $X = \{x_1, x_2, \ldots, x_n\}$ is the set of problem's variables.
- $D = \{D_1, D_2, \ldots, D_n\}$ is the domain value for each one of the variables.
- C is the constraint set imposed to the variables of the problem.

The *backtracking* family of algorithms [8,9] are used in practice to solve CSP problems. This kind of algorithms find the solution using a search tree. In this way a value is assigned to each variable for each node of the tree. There are several improvements available in the literature for the basic algorithms. Two of them are the MRV heuristic [10,7] and the FC method [11,7]. This two techniques are used joined to detect constraints inconsistencies earlier.

Minimum Remaining Values Heuristic (MRV) This heuristic looks to improve the execution time taking advantage of the selecting order of the problem's variables. The idea is to choose first the variables that have small domain value sets. Then quick search for inconsistency situations is performed for partial assignments. More details can be found in [10,7].

Forward Checking (FC) It is a method used to propagate information between the CSP constraints. When a variable X is assigned, the process takes every unassigned variable Y that is connected to X by a constraint and deletes all the values from the X´s domain that are inconsistent with the assigned value of X. More details can be found in [11,7].

In many real-life situations it is convenient to find the best solution of the admissible set. The quality of solution is usually measured by a function called Optimization or cost function.

The goal is to find the solution that satisfies all the constraints and minimize or maximize the optimization function respectively. Such problems are referred to as Constraint Satisfaction and Optimization Problems (CSOP) [12].

A CSOP consists of a standard CSP and an optimization function that maps every solution to a numerical value.

The most widely used algorithm for finding optimal solutions is called Branch and Bound (B&B) [13,12]. For further reading can be seen [14,15].

3 Distributed Computing Workflow Optimization

The different jobs that compose the workflow must be assigned to the available resources to execute them. Moreover constraints that come from hardware requirements have to be satisfied and workflow dependencies between jobs must to be accomplished. The problem can be encompassed in a Constrain Satisfaction Problem and the optimal solution has to be found among the different possible ones.

The objective of this paper is to propose a scheme that provides the optimal assignment of the jobs on the available resources satisfying the existing constraints. This optimal is the minimum execution time for the complete workflow. The number of admissible combinations growth in an exponential way with the quantity of jobs that compose the workflow. Consequently the problem can be considered as NP-Complete one.

The tool discussed in this work can be encompassed in the context of workflow scheduling problems. In this case some simplifications are assumed respect to job execution times and data transfer rates as well.

The architecture of the proposed scheme can be observed in figure 3 where the two main modules can be recognized: matchmaking and scheduling, that are organized in an uncoupled way. Matchmaking compares job execution requirements with available hardware resources. In this way if a particular job can not be executed the complete workflow is prevented to be processed. Then workflow scheduling has not merit in this scene. Matchmaking step can be obtained from middleware like Condor[16] or any other appropriate ad-hoc matchmaker.

After matchmaking takes place a list of available resources that satisfy execution requirements is provided for each job. This so called *matches list* together a proper performance data and workflow dependencies are used as input data for the scheduling module.

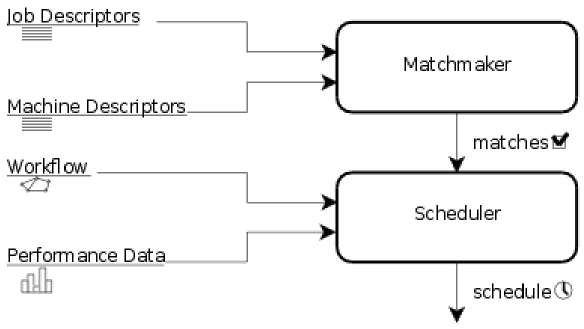


Figure 3. Process Overview.

The scheduler's diagram can be seen in figure 4, where the different components can be observed as well as their interactions.

At first place, variable, domain and constraint generators modules are executed. These modules uses as input data: i) Matching list between jobs and resources provided for matchmaking module; ii) Workflow dependencies description; iii) Performance data to be considered.

PERT (Program Evaluation and Review Technique) is a model developed by the US Army in the fifties. In this paper the application jobs of a workflow are modeled using PERT in order to measure the cost of the different mappings of the workflow over the resources. An example can be seen on figure 5. It represents a simple workflow of 6 tasks. Arcs represent tasks and nodes represent events (the beginning and the end of tasks). PERT plays a very important role in order to solve the CSP problem.

A CSP definition of the problem is obtained once the information provided by generator modules and PERT diagram is obtained from the workflow.

From the CSP definition and Cost performance model the optimizer module is executed in order to find the optimal solution. Backtracking and Branch and Bound algorithms are used in this case.

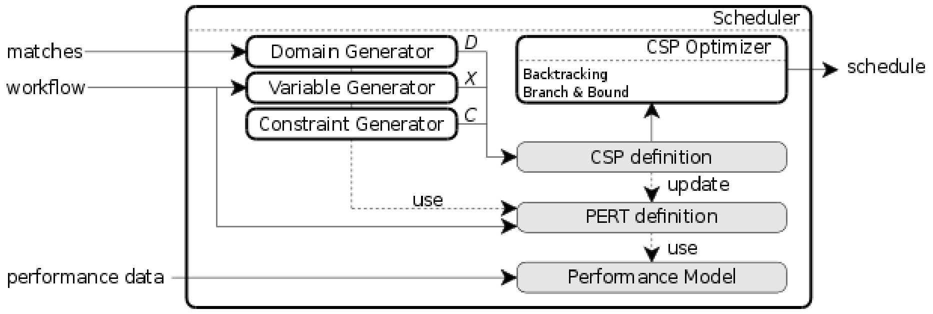
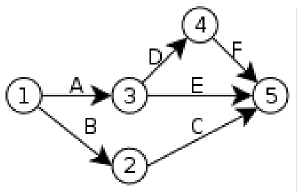


Figure 4. Scheduler Diagram.



Job	Cost	Sucessor Jobs
A	3	D, E
B	4	C
C	3	
D	1	F
E	2	
F	3	

Figure 5. PERT example.

3.1 Workflow Scheduling Problem Modeling

In order to obtain the mapping of the workflow onto the available resources the scheduling problem can be stated as an optimization problem. In this case the optimization or cost function is the total execution time of the problem.

According the ideas discussed in subsection 2.3 this problem can be modeled like a CSP:

– The set of variables is splitted in two different subsets. The variables $x_{j_i}, 1 \leq i \leq n$ denote workflow jobs and are called *job variables*. The variables $x_{r_i}, 1 \leq i \leq m$ belong to dependency relationships that could follow from graph independent jobs. These variables are called *dependency relationship variables*, and are denoted as *jobVariableX>>jobVariableY*.
– The domain of job variables are the resources (computers in this case) where jobs can be processed. The relationship variables domain take the values: *before*, *after* o *independence*. For instance *jobVariable1>>jobVariable2* points out that job 2 depends on job 1 when the value is *before*. For an *after* value job 2 depends on job 1 and *independence* means that both works can be processed independently.
– There are three different kinds of constraints:

- *PertCyclesConstraint*: this kind of restrictions control that no loops be created when new dependencies relationships are generated.
- *PertCostConstraint* is a soft constraint used to compute the partial or total execution time or cost problem.
- *JobOverlapConstraint* is a restriction that avoids overlapping time of independent jobs assigned to the same resource. In practice is computed as: $job1^+ < job2^- \vee job2^+ < job1^-$. Where job^- and job^+ denote time beginning and finalization jobs respectively, according to Qualitative Interval Constraints [17].

3.2 Performance Model

In order to measure the cost of solutions it is necessary to estimate the hardware performance. In the literature different approaches has been suggested and an overview of the subject can be found in [3]. In this work a simple *Performance Model* is used in order to estimate processing capacity and data transfer as well.

Execution Time The execution times are estimated for a given job in terms of processing capacity of resources per unit time and jobs processing requirements as can be seen in equation 1.

$$T_{ex}(jobX) = processingRequirement_{jobX}/processingCapacity_{machineX} \quad (1)$$

Where $processingRequirement_{jobX}$ denotes the processing requirement of $jobX$ measured in a proper unit (MIPS for instance) and the processing capacity of the assigned resource to $jobX$ is denoted as $processingCapacity_{machineX}$.

Transfer Time Transfer time are estimated in terms of message size and transfer rate in between the different resources assigned to the jobs as can be seen in equation 2.

$$T_{tx}(jobA, jobB) = message_{jobA}/transferRate_{machineA,machineB} \quad (2)$$

Where $message_{jobA}$ is the size of the message to be transmitted ($jobA$ output data) and $transferRate_{machineA,machineB}$ is the transfer rate between computers where jobs A and B are assigned respectively.

3.3 Optimizer

The proposed optimization algorithm is a version of *Backtracking* [8,7,9] algorithm combined with *Branch & Bound* [13,12]. This algorithm searches all possible solutions of the CSP and returns the best one according to the optimization function. This recursive algorithm is a deep-first search that assigns a value to each one of the different variables of the problem. It backtracks when all values have been tried for a variable and when the estimated solution cost of an assignment exceeds the cost of the best solution found at he moment.

In algorithm 1 can be seen the pseudocode of the optimization algorithm. The algorithm is invoked with an empty assignment (none of its variables has a value).

It starts checking if the assignment passed as parameter is complete (one in which all the variables have a value), in this case a *solution* was found and it is returned to the previous recursive steps of the algorithm (lines 5 and 6). If the assignment is not a solution, the algorithm selects a variable to assign (*selectUnassignedVariable* function on line 7). This is repeated on each recursive step. Then a domain value is selected to trial as new assignment. In most cases particular domain orderings conduces the algorithm to find solutions faster. The *orderDomain* function on line 8 must be implemented for this issue. In this version of the algorithm the default domain order is kept.

Once a variable and a value have been selected, the Forward Checking method is invoked (line 9) and the PERT is updated with the new assignment (line 10). Then, constraints must be checked in order to look for assignment inconsistencies (line 11). If all constraints are satisfied, the cost of the assignment must be computed (line 12). If the cost is lower than the minimum solution cost found at the moment (line 13), the assignment becomes effective (line 14). Then a new recursive step is initiated with the new assignment (line 15). The return value of each recursive invocation (each recursive step) may be a *solution* or a *null value*. *i)* If the result is a *solution*, it is kept (line 17) and its cost as well (line 18). After that, the last variable assignment is removed (line 19) for continue searching solutions. *ii)* If the result is a *null value* the algorithm removes the last variable assignment (line 19) this is made in order to perform a *backtrack*.

Once that all the values of a variable have been tried a *null value* is returned indicating that a *backtrack* must be performed (line 20).

Each new variable assignment generates a new workflow mapping over the resources with potential new dependency relationships. For each new mapping, cost constraint must be evaluated.

Algorithm 1 Scheduler Backtracking-Branch&Bound algorithm.

```
backtracking(assignment) returns an assigment
  global constraints //CSP constraints
  global solutionAssignment = null //solution
  global minimumCost = INFINITY //solution cost
  if (assignment.isComplete())
    return assignment //a solution was found
  variable = selectUnassignedVariable(assignment) //MRV
  for (value : orderDomain(variable))
    checkFoward(assignment, variable, value) //FC
    updatePert(assignment, variable, value)
    if (constraints.areSatisfied(assignment, variable, value))
      cost = constraints.cost() //for B&B
      if (cost < minimumCost)
        assignment.set(variable, value)
        assignment2 = backtracking(assignment)
        if (assignment2 != null)
          solutionAssignment = assignment2
          minimumCost = cost
        assignment.remove(variable)
  return null //all values tried, must backtrack

selectUnassignedVariable(assignment) returns a variable
  ... //implementation of MRV heuristic

orderDomain(variable) returns an ordered list of values
  ... //not implemented, the default order of domain is keeped
```

The optimization function is computed as the cost of *CostConstraint*. This function is invoked for each new assignment during the execution of the backtracking algorithm. In this way the algorithm can discard search-subtrees that exceeds in cost the best solution found. This step is known as *pruning*. The algorithm requires a proper heuristic function in order to have an estimation of the solution cost from incomplete assignments.

The optimizer founds an optimal or an approximated solution depending on the heuristic function used for performance estimation of workflow mappings.

In order to guarantee the algorithm's *optimality* such heuristic must be admissible, this is, an heuristic that don't overestimates the cost of the best solution. The use of non admissible heuristics (those that could overestimates the real cost of the best solution) leads us to an *approximation* problem.

3.4 Optimization Problem

The optimization function is calculated as $f = PertCostConstraint.cost$, this is the execution time for the complete workflow. This calculation is made by the evaluation of the PERT definition.

PERT is evaluated according the following rules:

- Execution time of a job depends on the machine on where is executed. This information is retrieved by the Performance Model. This could be seen on section 3.2.
- Transfer time of a message between two jobs depends on the message size and the transference rate between this two machines. This information is also retrieved by the Performance Model. As can be seen on subsection 3.2.
- Start time of a job B is the maximum ending time of its predecessor jobs. The ending time of a job A is calculated as: start time of A plus execution time of A plus transfer time of the output of A (message) from machine A to machine B. If a job has no predecessors, its start time is 0.
- The total cost of PERT is the maximum ending time of the final jobs of the workflow. A final job has no successors.

The admissible heuristic function consists on:

- Suppose that a non assigned job to a resource is executed on a machine with the maximum processing capacity. This estimation is the exact when there is only one possible machine for that job or all machines have the same processing capacity, like usually is the case of Beowulf Clusters in practice.
- Suppose that a data transfer between two jobs, in which at least one of them has no machine assigned, has the same cost as the two tasks were on the same machine (a local transfer).

3.5 Approximation Problem

In optimization problems the full solution space has to be searched in order to find the best one. Usually this task has a very high cost in terms of computational effort of is unacceptable in some cases. Then an *approximation* problem can be solved with a cost function close enough to the optimal one. The idea is to find an approximate solution for the workflow mapping provided approximation error is bounded. In this case near optimal workflow mapping can be computed with a considerably less computational effort than the optimization problem. In order to obtain an approximation problem from the optimization one a non admissible heuristic is chosen in order to estimate the solution cost for incomplete assignments. This kind of heuristic usually overestimate the solution cost respect the optimal one.

The proposed non admissible heuristic is based on the following ideas:

- Unassigned jobs are processed on average capacity resources. It is immediate to see that this heuristic is exact for some particular cases: there is only one possible machine for that job or all machines have the same processing capacity.

– It is assumed that messages between two jobs are transmitted with average transfer rate when at least one of the jobs have not a resource assigned.

4 Experiments

In order to test the proposed algorithm, different problems are solved. Architectures like Supercomputers, Beowulf Clusters and Condor Pools are considered. On this infrastructures a simple workflow is mapped.

4.1 Workflow

A simple workflow shown in figure 6, is chosen. Different problems denoted *Computing Intensive* and *Data Intensive* are defined. In the first case, high performance capacity is desirable and for the second case a good data transfer rate is a requisite.

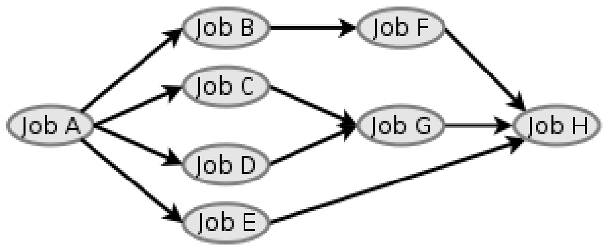


Figure 6. Eight-Jobs workflow.

Processing requisites and output sizes can be seen in tables 1 and 2 for Computing Intensive and Data Intensive simulations, respectively. Processing requirements are measured in *processing units* and output sizes are measured in Mib.

Table 1. Eight-Jobs Data Intensive problem configuration.

Job	A	B	C	D	E	F	G	H
processingRequirement	5,0e+5	4,0e+5	7,0e+5	1,0e+5	2,0e+5	1,0e+5	3,0e+5	2,0e+5
Output size (*message*)	1,0e+3	2,5e+3	2,0e+3	1,5e+3	3,0e+3	1,7e+3	2,3e+3	0

Table 2. Eight-Jobs Computing Intensive problem configuration.

Job	A	B	C	D	E	F	G	H
$processingRequirement$	5,0e+6	4,0e+6	7,0e+6	1,0e+6	2,0e+6	1,0e+6	3,0e+6	2,0e+6
Output size ($message$)	100	19	77	120	91	60	149	0

4.2 Hardware Infrastructure

In this subsection processing capacity and transfer rate is discussed for the different architectures considered.

Supercomputer This type of architecture is often used in high performance computing in order to solve Bioinformatics applications, Weather simulation, Computational Fluid Dynamics, etc.

In this case four processor are considered with a $processingCapacity = 500$ (processing units / second) for all of them. In order to define data transfer rate, different works running on the same processor and different processors cases must be accounted, as can be seen in table 3.

Table 3. Supercomputer transfer rates in Mib/s.

Case	$transferRate_{X,Y}$
$X \neq Y$	10000
$X = Y$	200000

Beowulf Cluster In this case an homogeneous cluster is defined with four nodes with a $processingCapacity = 100$ (processing units / second) each one. Transfer rate between jobs processed in different nodes and in the same node can be seen in table 4.

Table 4. Beowulf Cluster transfer rates in Mib/s.

Case	$transferRate_{X,Y}$
$X \neq Y$	1000
$X = Y$	10000

Transfer rate in this case is considered like a Gigabit Ethernet infrastructure and is lower than data transfer performance of supercomputer.

Condor Pool In this case partially available resources linked through a Fast Ethernet network with data $transferRate = 100$ are considered. For two jobs processed on the same machine $transferRate = 10000$, like the Beowulf Cluster, are selected. Both transfer rates are measured in Mib/s.

In this case the four nodes in the pool have different processing capacity (processing units / second) as can be seen in table 5.

Table 5. Condor Pool processing capacities in processing units / second.

Machine	1	2	3	4
$processingCapacity$	70	100	95	97

4.3 Discussion of Results

Optimal Solutions On table 6 can be seen results of the optimization problems. Column SC_O denotes the cost of optimal solution; VA_O denotes the number of variable assignments required to find the solution, this is the number of times that a value was selected for a variable; T_O show the execution time (in milliseconds) of the scheduler required in order to find the solution .

Table 6. Results for Optimization Problem.

Problem	SC_O	VA_O	T_O
Data Intensive, Supercomputer	3300.6	132573	10317
Computing Intensive, Supercomputer	33000.0	84297	6049
Data Intensive, Cluster	17003.1	132429	9513
Computing Intensive, Cluster	170000.1	85545	6114
Data Intensive, Condor Pool	18130.8	25260	2771
Computing Intensive, Condor Pool	180930.5	25260	2822

From table 6 can be seen that: as the number of generated assignments increase, the computation time of the algorithm increases as well. It is more difficult for the algorithm to find solutions when the architecture is homogeneous. This is because there are many solutions that have equivalent costs, and the algorithm checks all of them. An special treatment must be taken in account for this type of architectures.

Also can be mentioned that the solution costs of Data Intensive and Computing Intensive problems are similar but with one more magnitude order for the second one. This is because the processing requirements of the jobs in the Computing Intensive problem has one more magnitude order than the requirements of the Data Intensive problem.

For the Condor Pool case a much smaller number of variable assignments were necessary. This is because in heterogeneous architecture the algorithm can discard partial solutions earlier for suboptimal variable assignments. Consequently the solution can be found more easily than in the previous cases.

Approximate Solutions Table 7 show results of the approximation problems. Column SC_A show the cost of approximate solution; VA_A is the number of variable assignments required to find the approximate solution; T_A show the execution time (in milliseconds) of the scheduler to find the approximate solution.

Table 7. Results for Approximation Problem.

Problem	SC_A	VA_A	T_A
Data Intensive, Supercomputer	3400.3	9978	1262
Computing Intensive, Supercomputer	34000.0	9811	832
Data Intensive, Cluster	17000.5	9978	769
Computing Intensive, Cluster	170000.0	9811	730
Data Intensive, Condor Pool	17350.5	1507	193
Computing Intensive, Condor Pool	173500.0	1507	217

A different behaviour can be seen respect to the optimization problem cases. For the approximation problem the variable assignments produced by the algorithm are very similar, except on Condor Pool architecture mappings. That continue making a much smaller number of variable assignments on that architecture type than the others.

Comparison of results for optimal and approximate solutions The comparative results of approximation and optimization problems can be seen on table 8. Column $SC_{Err} = 100\% * |SC_O - SC_A| / SC_O$ show relative error between solution costs of optimization and approximation problems. Column VA_O/VA_A show ratios of the number of variable assignments between optimal and approximate solutions. Column T_O/T_A show scheduling time ratios between optimal and approximate solutions.

Table 8. Comparison results.

Problem	SC_{Err}	VA_O/VA_A	T_O/T_A
Data Intensive, Supercomputer	0.01%	13.29	8.,18
Computing Intensive, Supercomputer	0.00%	8.59	7.27
Data Intensive, Cluster	0.02%	13.27	12.37
Computing Intensive, Cluster	0.00%	8.72	8.38
Data Intensive, Condor Pool	4.50%	16.76	14.36
Computing Intensive, Condor Pool	4.28%	16.76	13.00

Column T_O/T_A in table 8 show that execution times of the algorithm reduces significantly. About 7 times in worst case and 14 times in best case. A more precise measurement is the comparison between variable assignments (VA_O/VA_A), because only involves the characteristics of the algorithm execution and do not have into account issues related to the Operating System, I/O's, another running processes, etc. This metric show that in the approximation problems a reduction of the assignments is about 16 times in best case and 8 times in worst case. For the cases in which $VA_O/VA_A = 16.76$ a $1 - 1/16.76 = 94.03\%$ of the assignments made in optimization problem were not made on the approximation approach. That is why the relative error range a 4%. Also can be seen that Data intensive problems on Supercomputer and Cluster architectures have the best relation (*ratio*) between approximation of optimal solution and algorithm execution time. This is because in homogeneous architectures the execution times of the jobs are known a priori without knowing the specific machine in which the job will be executed. In this way the execution times estimation made by the heuristics is exact.

5 Concluding Remarks

- In order to validate the proposed algorithm some experiments have been carried out . A workflow has been processed on different (simulated) distributed computing architectures: Supercomputer, Beowulf Cluster and Condor Pool.
- The algorithm have found optimal solution for all the architectures tested. However some difficulties were found for homogeneous resources like Supercomputers and Cluster instead of a good behaviour obtained for the Condor Pool.
- In general approximate solutions have very good quality and have been obtained with fairly less computational effort, than the optimal case.
- From the obtained results can be said that the design of proper heuristics is a central point in order to obtain a good at a reasonable execution time for the workflow scheduling.
- It appears to be important to design a new heuristic for homogeneous case. Some improvements for the algorithms can be studied as well. The idea is to implement ordering of domain values. On the other hand Arc Consistency can be studied instead of Forward Checking.

References

1. Jack Dongarra, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon, and Andy White, editors. *Sourcebook of parallel computing.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
2. Joseph Sloan. *High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI (Nutshell Handbooks).* O'Reilly Media, Inc., November 2004.
3. Jia Yu and Rajkumar Buyya. A taxonomy of workflow management systems for grid computing, Apr 2005.
4. Vlado Stankovski, Martin Swain, Valentin Kravtsov, Thomas Niessen, Dennis Wegener, Jörg Kindermann, and Werner Dubitzky. Grid-enabling data mining applications with datamininggrid: An architectural perspective. *Future Gener. Comput. Syst.*, 24(4):259–279, 2008.

5. Abbas A. Grid Computing: A practical Guide to Technology and Applications. *Charles River Media*, 2003.
6. Foster I. and Kesselman C. (eds.). The Grid: Blueprint for a New Computing Infrastructure. *Morgan Kaufmann*, 1999.
7. Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice-Hall, 2002.
8. James R. Bitner and Edward M. Reingold. Backtrack programming techniques. *Commun. ACM*, 18(11):651–656, 1975.
9. Gilles Brassard and Paul Bratley. *Fundamentals of Algorithmics*. Prentice-Hall, 1995.
10. Fahiem Bacchus and Paul van Run. Dynamic variable ordering in csps. In *CP '95: Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pages 258–275, London, UK, 1995. Springer-Verlag.
11. Fahiem Bacchus and Adam J. Grove. On the forward checking algorithm. In *CP '95: Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pages 292–308, London, UK, 1995. Springer-Verlag.
12. Roman Bartak. Constraint programming: In pursuit of the holy grail. In *in Proceedings of WDS99 (invited lecture*, pages 555–564, 1999.
13. E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.
14. Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 266–271, Edinburgh, Scotland, August 2005.
15. Adrian Petcu and Boi Faltings. ODPOP: An algorithm for open/distributed constraint optimization. In *Proceedings of the National Conference on Artificial Intelligence, AAAI-06*, pages 703–708, Boston, USA, July 2006.
16. *Condor: www.cs.wisc.edu/condor/.*
17. Eddie Schwalb and Lluís Vila. Temporal constraints: A survey. *Constraints*, 3(2/3):129–149, 1998.